REPORTS IN INFORMATICS

ISSN 0333-3590

A completely dynamic algorithm for split graphs

Pinar Heggernes **Federico Mancini**

REPORT NO 334 **November 2006**

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL http://www.ii.uib.no/publikasjoner/texrap/ps/2006-334.ps

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at http://www.ii.uib.no/publikasjoner/texrap/.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

A completely dynamic algorithm for split graphs*

Pinar Heggernes† Federico Mancini†

Abstract

We present a fully dynamic algorithm for split graphs that supports the following types of operations: (1) query whether deleting or inserting an edge preserves the split property, (2) query whether inserting a new vertex with a given neighborhood in the current graph preserves the split property, (3) insert or delete an edge or a vertex when the split property is preserved, (4) insert an edge or a vertex even when the split property is not preserved, adding a minimal or a minimum set of additional edges to keep the graph split, (5) delete an edge even when the split property is not preserved, deleting a minimal or a minimum set of edges from the current graph to keep the graph split. Usually, fully dynamic algorithms for recognizing and maintaining a class of graphs support operations of type (1), (2), and (3). Because of the additional operations we call our algorithm *completely dynamic*.

As an interesting consequence, we show that the above operations very easily lead to the following results, in addition to the dynamic algorithm: a linear-time vertex incremental certifying algorithm for recognizing split graphs, and a linear-time vertex incremental algorithm for computing a minimal split completion of an arbitrary graph. Both of these algorithms match the best known algorithms for these purposes, but have the advantage og being vertex incremental thusly allowing the input to be given on-line.

Finally, by the dynamic operations of type (4) and (5), we show that the following problems can be solved in linear time: adding a minimum number of edges to a given split$+1v$ or split$+1e$ graph to obtain a split graph, and removing a minimum number of edges from a given split$-1e$ graph to obtain a split graph.

1 Introduction

Dynamic graph algorithms allow incremental updates to a given graph and at the same time preserve a given graph property faster than checking the property on the whole updated graph. Such algorithms are called *fully dynamic* if they allow both insertion and deletion operations. In particular fully dynamic algorithms for recognizing and maintaining a graph class are able to check whether the input graph is still in the class after the deletion or addition of an edge or a vertex (a *query* operation), and update the graph accordingly if the answer to corresponding query is yes (an *update* operation), efficiently. Fully dynamic algorithms have been developed for recognizing and maintaining cographs [17], permutation graphs [2], interval graphs [13], proper interval graphs [10], chordal graphs [11], and split graphs [12].

Split graphs are a well-studied graph class with wide theoretical use [4], and they can be recognized in time $O(n + m)$ for input graphs with n vertices and m edges [4, 5]. A dynamic algorithm for split graphs, supporting the following operations of type (1) and (3) was given by Ibarra [11, 12]: query for edge addition and deletion, update after deleting or adding an edge when the answer to the corresponding query is yes. His algorithm is based on the characterization of split graphs by their degree sequences [5], and each query or update for edge addition or edge deletion takes constant time, after an initial linear-time preprocessing. What we present in this paper is a fully dynamic algorithm for split graphs, but it is based on a more powerful structure: the 3-partition defined recently by the authors [6]. This structure allows us to support the following additional operations: query for addition of a new vertex with a given neighborhood in the current graph, update after adding a vertex when this addition preserves

*This work is supported by the Research Council of Norway through grant 166429/V30.

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Emails: pinar@ii.uib.no and federico@ii.uib.no

the split property, update after deleting a vertex and all its incident edges (this always preserves the split property), update after adding a vertex when this addition does not preserve the split property, update after adding or deleting an edge when this addition or deletion does not preserve the split property. In the last two types of update operations, for addition, the user has the choice between a minimal or a minimum set of edges that are added to the graph to preserve the split property, in addition to the desired edge(s). For deletion, again there is the choice between a minimal or a minimum set of edges of the current graph that are deleted in addition to the chosen edge, to preserve the split property. Because of these additional operations, we call our algorithm a *completely dynamic* algorithm.

In the dynamic algorithm of Ibarra [12], each query and update operation takes $O(1)$ time for edge addition or deletion. Our algorithm is able to match this time bound for each of these operations.

When it comes to the new dynamic operations presented in this paper, a query of whether a new vertex x, with a given neighborhood N_x in the current graph, can be added without destroying the split property is replied in time $O(|N_x|)$, and if the answer is yes, the corresponding update also takes $O(|N_x|)$ time. When the reply is no, the minimal update operation adds x to the graph making it adjacent to N_x and to an inclusion minimal set of additional vertices A_x, and this takes $O(|N_x| + |A_x|)$ time. In case the reply to a query for the addition of an edge uv is no, the corresponding minimal update operation adds either an inclusion minimal set of edges A_u incident to u or an inclusion minimal set of edges A_v incident to v, in addition to uv, and the time required for this update is either $O(d(u) + |A_u|)$ or $O(d(v) + |A_v|)$, according to which endpoint we choose. In case of the reply to a query for the deletion of an edge uv is no, the corresponding minimal update operation removes an inclusion minimal set of edges from one of the endpoints of uv, and the time required for this update is $O(d(v))$ or $O(d(u))$, according to which endpoint we chose.

The dynamic operations of adding a new vertex either when the graph remains split or when we need to add edges to keep it split, very easily lead to following two new vertex incremental algorithms: A certifying algorithm that recognizes split graphs in time linear in the size of the input graph, and an algorithm for computing a minimal split completion of an arbitrary graph in time linear in the size of the output graph. Note that for both problems, algorithms exist for solving them within these time bounds [8, 6], however none of the existing algorithms are vertex incremental. The resulting new vertex incremental algorithms for these problems allow them to be solved in an on-line fashion, where the input graph is supplied one vertex at a time.

Finally, our dynamic algorithm also contains operations for updating the graph by adding or deleting a *minimum* set of additional edges when the reply to a query is no, but the update is still desired. In particular, with these operations we are able to show that the following problems can be solved in linear time: Computing minimum split completions of split$+1v$ and split$+1e$ graphs, and computing minimum split deletions of split$-1e$ graphs. Similar results have been shown for cographs [16]. To show these results we first give linear-time certifying recognition algorithms for these classes.

In the next section, we introduce some notation and list some results from [6] that will be used in this paper. The new dynamic algorithm is presented in Section 3, where we give some new lemmas to decide under which conditions a split graph remains split after the addition or deletion of an edge or a vertex, and we describe in detail all operations supported by the new dynamic algorithm. Sections 4 and 5 are dedicated to explaining spin-off results based on the new dynamic operations: we present the above mentioned vertex incremental algorithms and the algorithms for split$+1v$, split$-1e$, and split$+1e$ graphs. Finally we give some concluding remarks in section 6.

2 Definitions and background

All graphs in this paper are simple and undirected. For a graph $G = (V, E)$, we let $n = |V|$ and $m = |E|$. We will also use $V(G)$ and $E(G)$ to denote the vertex and edge sets of a graph, respectively. The set of *neighbors*, or the *neighborhood* of a vertex $v \in V$ is denoted by $N_G(v) = \{u \mid uv \in E\}$, or simply $N(v)$ if there is no ambiguity, and the *degree* of v is $d(v) = |N(v)|$. The neighborhood of a set of vertices $S \subseteq V$ is defined as $N(S) = \bigcup_{x \in S} N(x) \setminus S$. A vertex is called *universal* if it is adjacent to all other vertices in

the graph, and it is called *isolated* if it has no neighbors. An *induced subgraph* of $G = (V, E)$ over a set of vertices $U \subseteq V$, is the graph $G[U] = (U, E_U)$, where $E_U = \{uv \in E \mid u, v \in U\}$. The *complement* $\bar{G}$ of G consists of all vertices of G and $uv \in E(\bar{G})$ if and only if $uv \notin E(G)$.

A set of vertices $K \subseteq V$ is a *clique* if $G[K]$ is complete. A set of vertices $I \subseteq V$ is an *independent set* if no two vertices of I are adjacent in G. We use $\omega(G)$ to denote the size of a largest clique in G, and $\alpha(G)$ to denote the size of a largest independent set in G. Although computing $\alpha(G)$ and $\omega(G)$ are NP-hard problems for an arbitrary graph G, when G is a split graph $\alpha(G)$ and $\omega(G)$ can be computed in linear time [4, 5].

A graph $G = (V, E)$ is a *split graph* if there is a partition $V = I + K$ of its vertex set into an independent set I and a clique K. Such a partition is called a *split partition* of G. Split graphs can be recognized and a split partition can be found in linear time [5]. A split partition is not necessarily unique. The following theorem from [5] states the possible partition configurations.

Theorem 1 (Hammer and Simeone [5]) *Let $G = (V, E)$ be a split graph with a split partition $V = I + K$. Exactly one of the following conditions holds:*

1. *$|I| = \alpha(G)$ and $|K| = \omega(G)$ (in this case the partition $I + K$ is unique),*
2. *$|I| = \alpha(G)$ and $|K| = \omega(G)-1$ (in this case there exists a vertex $x \in I$ such that $K \cup \{x\}$ is a clique),*
3. *$|I| = \alpha(G)-1$ and $|K| = \omega(G)$ (in this case there exists a vertex $y \in K$ such that $I \cup \{y\}$ is independent).*

Split graphs are hereditary, which means that every induced subgraph of a split graph is also a split graph [4]. For the following result, note that a simple cycle on k vertices is denoted by C_k and that a complete graph on k vertices is denoted by K_k. Thus $2K_2$ is the graph that consists of 2 isolated edges.

Theorem 2 (Földes and Hammer [3]) *Let G be an undirected graph. The following conditions are equivalent:*

1. *G is a split graph.*
2. *G and $\bar{G}$ are chordal graphs.*
3. *G contains no induced subgraph isomorphic to $2K_2, C_4$ or C_5.*

For a given arbitrary graph $G = (V, E)$, a split graph $H = (V, E \cup F)$, with $E \cap F = \emptyset$, is called a *split completion* of G. The edges in F are called *fill edges*. H is a *minimum* split completion of G if there is no set F' with $|F'| < |F|$ such that $(V, E \cup F')$ is a split graph. Minimum split completions are NP-hard to compute [15]. H is a *minimal* split completion of G if $(V, E \cup F')$ fails to be a split graph for every proper subset F' of F. Minimal split completions can be computed in linear time [6]. A *minimal split deletion* is an edge-maximal split subraph on the same vertex set.

A graph is *split*$+1v$ (*split*$+1e$) if it contains a vertex (edge) whose deletion from the graph results in a split graph. A graph is *split*$-1e$ if a split graph can be obtained by adding an edge to it.

Throughout the paper we will be using results from [6], most of which are based on a new kind of partition for split graphs. A 3-*partition* of the vertices V of a split graph $G = (V, E)$ consists of three sets $V = S + C + Q$ defined as follows:

$$S = \{v \in V \mid d(v) < \omega(G) - 1\} \qquad C = \{v \in V \mid d(v) > \omega(G) - 1\} \qquad Q = \{v \in V \mid d(v) = \omega(G) - 1\}$$

This partition is obviously unique, however it has been defined only to overcome the ambiguity caused by the many possible split partitions a split graph can have; therefore, when a graph has a unique split partition $V = I + K$, we define the 3-partition as follows:

$$S = I = \{v \in V \mid d(v) \leq \omega(G) - 1\} \qquad C = K = \{v \in V \mid d(v) > \omega(G) - 1\} \qquad Q = \emptyset$$

The uniqueness of a split partition can be checked by Theorem 1 and by the following lemma, hence it is easy to maintain the 3-partition correctly. In our dynamic algorithm, we will have a global variable that indicates at all times whether or not the current graph has a unique split partition, thus Q will be empty exactly when the graph has a unique split partition.

Lemma 3 ([6]) *A split graph $G = (V, E)$ has a unique partition $V = I + K$ if and only if there are exactly $\omega(G)$ vertices of degree greater than $\omega(G) - 1$.*

The following properties of the 3-partition will be useful in this paper.

Property 4 ([6]) *Given a split graph $G = (V, E)$ and its 3-partition $V = S + C + Q$, any split partition $V = I + K$ of G satisfies $S \subseteq I$ and $C \subseteq K$.*

Property 5 ([6]) *Let $G = (V, E)$ be a split graph with 3-partition $V = S + C + Q$. Every vertex of Q is adjacent to all vertices of C and to no vertex of S.*

Lemma 6 ([6]) *Let $G = (V, E)$ be a split graph with 3-partition $V = S + C + Q$. One of the following is true:*

1. *Q is a clique and $|C| + |Q| = \omega(G)$.*
2. *Q is an independent set, $|C| = \omega(G) - 1$, and $|Q| \geq 2$.*

Property 7 *If a split graph $G = (V, E)$ with 3-partition $V = S + C + Q$ has a unique split partition or Q is a clique, then each vertex in C has at least one neighbor in S.*

Proof. It has been proved in [6] that this is true when G has a unique partition. When Q is a clique, $Q \cup C$ is the maximum clique of the graph by lemma 6, so let us assume for the sake of contradiction that a vertex $c \in C$ does not have any neighbor in S. This means that all neighbors of c are in $C \cup Q$, hence $d(c) = \omega - 1$, contradicting the fact the it belongs to C by the definition of 3-partition. ∎

Notice that when there is only one vertex in Q, Q is both an independent set and a clique, but we assume it to be a clique unless we specify differently. The following theorem characterizes minimal split completions, and will be useful for the update operations when the reply to the corresponding query is no.

Theorem 8 ([6]) *Let $H = (V, E + F)$ be a split completion of an arbitrary graph $G = (V, E)$, and let $V = S + C + Q$ be the 3-partition of H. H is a minimal split completion of G if and only if each fill edge has both its endpoints in C.*

3 A dynamic algorithm for maintaining split graphs

Our dynamic algorithm starts by taking an input graph (which might be empty), checking that it is split, and setting up the data structure for its 3-partition, which can be done in linear time [6]. At all steps, we will have both the current graph and its 3-partition correctly stored. Thus for each dynamic operation, we assume that the input graph for this operation is split, and that we have the correctly stored 3-partition of it available. Before we explain the data structure details, we find it useful to formalize how the 3-partition of a split graph and the 3-partition of its complement are related. The following lemma and corollary will be useful when working with edge deletions, since they are equivalent to edge additions to the complement graph.

Lemma 9 *Let $G = (V, E)$ be a split graph with 3-partition $V = S + C + Q$ and let $\bar{G}$ be the complement of G, with $\omega = \omega(G)$ and $\bar{\omega} = \omega(\bar{G})$.*

1. *For each vertex $s \in S$, $d_{\bar{G}}(s) > \bar{\omega} - 1$.*
2. *For each vertex $c \in C$, $d_{\bar{G}}(c) \leq \bar{\omega} - 1$ if $|C| = \omega$, and $d_{\bar{G}}(c) < \bar{\omega} - 1$ otherwise.*

3. For each vertex $q \in Q$, $d_{\bar{G}}(q) = \bar{\omega} - 1$.

Proof. Since G is a a split graph, we know that $\bar{G}$ is also a split graph and $\alpha(G) = \omega(\bar{G})$. If Q is an independent set or empty then $\alpha(G) = |S| + |Q|$, and thus the maximum clique of $\bar{G}$ is $Q \cup S$ and is unique. If Q is a clique then $\alpha(G) = |S| + 1$, and for each $q \in Q$, $S \cup \{q\}$ is a maximum clique in $\bar{G}$. This also means that no vertices of C are in a maximum clique of $\bar{G}$.

A vertex $s \in S$ belongs to any maximum clique in $\bar{G}$, meaning that $d_{\bar{G}}(s) \geq \bar{\omega} - 1$. If Q is an independent set of any size or empty in G, no vertex of $s \in S$ can be adjacent to every vertex of C in G. This means that in $\bar{G}$, s is adjacent to at least one vertex that is outside a maximum clique, implying $d_{\bar{G}}(s) > \bar{\omega} - 1$. If Q is a clique of size least 2 in G, we again have $d_{\bar{G}}(s) > \bar{\omega} - 1$, because s is in the intersection of at least two maximum cliques of $\bar{G}$.

A vertex $c \in C$ is adjacent in $\bar{G}$ to only those vertices of S that it is not adjacent to in G. Now, if $|C| = \omega$, then $Q = \emptyset$ and c has at least one neighbor in S by Property 7. Therefore, since $|S| = \omega(\bar{G})$ then $d_{\bar{G}}(c) \leq \bar{\omega} - 1$. When $|C| < \omega$, we have to consider Q. If Q is an independent set of size at least 2, c might not have any neighbor in S, in which case $d_{\bar{G}}(c) \leq |S|$, but $\omega(\bar{G}) = |S| + |Q| \geq |S| + 2$, so $d_{\bar{G}}(c) < \bar{\omega} - 1$. If Q is a clique, instead, $\omega(\bar{G}) = |S| + 1$ and c must have at least a neighbor in S by Property 7, meaning that $d_{\bar{G}}(c) \leq |S| - 1$, so again $d_{\bar{G}}(c) < \bar{\omega} - 1$.

A vertex $q \in Q$ always belongs to a maximum clique in $\bar{G}$ by the observation we made in the beginning. Thus, either $S \cup \{q\}$ is a maximum clique in $\bar{G}$, meaning that q is adjacent to nothing else than S in $\bar{G}$ because $C \cup Q$ is a clique in G, or $Q \cup S$ is a maximum clique in $\bar{G}$, and q is not adjacent to any vertex of C in $\bar{G}$ because it is adjacent to all of them in G. In either case $d_{\bar{G}}(q) = \bar{\omega} - 1$. ∎

Corollary 10 *Let $G = (V, E)$ be a split graph with 3-partition $V = S + C + Q$. Then $V = \bar{S} + \bar{C} + Q$ is the 3-partition of $\bar{G}$ with $\bar{S} = C$ and $\bar{C} = S$.*

Proof. Follows by Lemma 9 and the definition of 3-partition. ∎

3.1 The data structure

In this section we describe how to implement the 3-partition as a data structure for our dynamic algorithm, and we give the details needed to justify the running time of the operations that will be analyzed in the next sections.

Given an input graph G represented with an adjacency list, as an initial preprocessing step, before the dynamic algorithm begins, we can find the degrees of all vertices in linear time. Using the degrees, $\omega(G)$ can be found in linear time, as well [5]. After this, the vertices can be partitioned into S, C, and Q according to their degrees, in linear time. The sets S and C, are each represented by an object containing a degree array of n elements where each element d of the array has two pointers: one to an object containing the label of the set, and one to a list of elements representing the vertices in that set with degree d. We call such a list, a degree list L_d. Each element in a degree list points both to the object with the label of the set it belongs to, and to the corresponding vertex of the graph. There can therefore be up to n labels for a set, but only one for each degree list (see Figure 3.1 for an example). Besides, we keep a pointer to the maximum and minimum non empty element of the degree array, and, from the last element of a degree list, we have a double pointer to the first element of its and the next degree list in the array.

Notice also that, since in Q all vertices have always the same degree, we do not need a degree array, but just a simple list of elements representing the vertices in Q and a label to which they all point. However we maintain one more variable for Q that holds a value from $\{0, 1, 2\}$, indicating, respectively, whether Q is empty, a clique, or an independent set (of size at least 2, or we consider it a clique). This information can be updated and accessed in constant time.

Eventually, we also maintain a variable with the size of the maximum clique of the graph and each vertex of the graph has a double pointer to its corresponding element in a degree list.

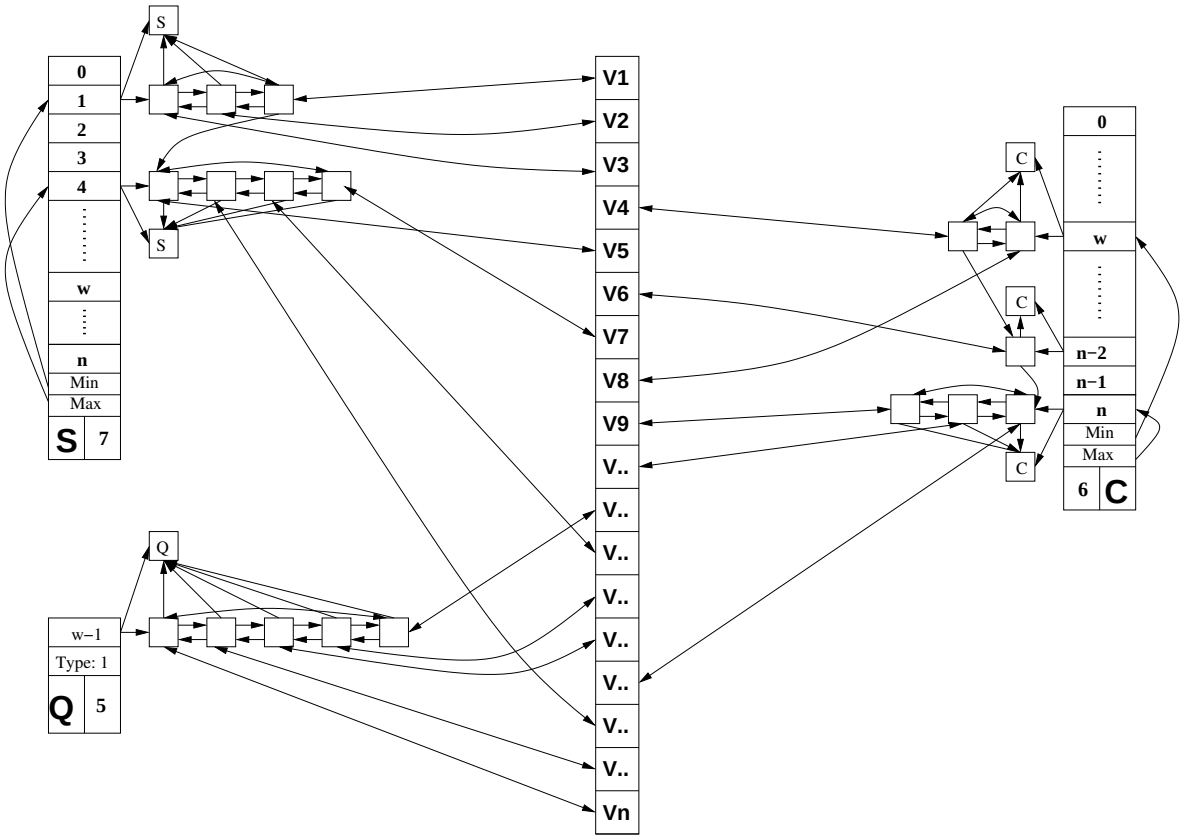

Figure 1: In this figure we give an example of the data structure described in section 3.1. We have the adjacency list of our graph in the middle, and the three objects representing the sets of the 3-partition. S and C contain each a degree array of size n, and their non-empty elements point to the corresponding degree list. Each degree list points to the next one, and each element in the list points to both its label and the vertex it represents. Finally for each set we store its size and two pointers to the maximum and minimum non empty element of its degree array. For Q we have only a degree list corresponding to the vertices of degree $\omega(G) - 1$, and we keep two variables: one with the type of Q, and another with its size.

This way, finding out which set a given vertex belongs to or its degree, deleting a given vertex from a set, and adding a given vertex to a set take each constant time. Consequently, moving a given set A of vertices, takes at most $O(|A|)$ time.

Given this structure, to scan or return a set of the 3-partition takes time linear in the size of the set, because the degree lists are all connected by pointers, so we can just scan them one after the other, starting with the one corresponding to the minimum degree in the array, to which we have a pointer as well.

As we said, the size of a degree array is n, but since we allow addition of vertices to the graph, for every such operation we should increase the size of the array by 1. This can be done in amortized constant time using dynamic arrays, or allocating from the beginning an array of size $n + n'$, where n' is an upper bound to the number of possible vertex additions given by the user of the algorithm. Notice that this is something every dynamic algorithm has to deal with, independently from the 3-partition.

Now we give a lemma about the running time of some specific operations on this data structure, that will be useful when analyzing the running time of the algorithms in the next sections.

Lemma 11 *Let $X, Y \in \{S, C, Q\}$ for the 3-partition $V = C + S + Q$ of a split graph $G = (V, E)$, and let $A \subset X$. We can move all vertices $X_d = \{x \mid x \in X \setminus A \wedge d(x) = d\}$ to Y in time $O(\max\{|A|, 1\})$, if Y has a constant number of vertices of degree d.*

Proof. The idea of the proof is the following: first we take out from L_d of X the elements of A of degree d, and we create a new list L_d^A with them in $O(|A|)$ time; then we move in constant time what is left of L_d of X, namely X_d, to Y; and eventually we make L_d^A the new list L_d of X in constant time.

All vertices of X_d have the same degree d, and therefore belong to the degree list L_d of X. If there are some vertices of A in L_d, we delete them from L_d and we create a new degree list L_d^A consisting only of such vertices. Then we create a new label X and we move the pointers of all elements of L_d^A to it. To find such vertices of A and perform all the operations described takes time $O(|A|)$. After we remove the elements of A from L_d, notice that what is left is exactly X_d.

Since we assumed that there is only a constant number of vertices with degree d in Y, after we changed the label to which X_d still points, from X to Y, and we appended X_d to L_d of Y, we can move in constant time the pointers of the elements in L_d of Y, from the label they point to (that we will delete), to the label of X_d. To enqueue X_d to L_d of Y takes constant time, and so it does updating the pointers from and to the new L_d of Y.

Eventually we move the pointers of the element d of the degree array of X to L_d^A and its label, and update the other pointers of the degree array of X, in $O(1)$ time.

Therefore the overall running time to move X_d to a set Y and update the current set X, is $O(\max\{|A|, 1\})$, since if $A = \emptyset$, $L_d = X_d$ and we can move it to Y in constant time. ∎

3.2 Adding a vertex with a given neighborhood in G

In this section, we describe the query and update operations for adding a new vertex x with a given desired neighborhood $N_x \subseteq V$ to a split graph $G = (V, E)$. We define $G_x = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$. A query for adding vertex x is then simply deciding whether G_x is a split graph. This operation can be implemented in time $O(|V| + |E| + |N_x|) = O(|V| + |E|)$ by using the fastest split recognition algorithms. We will show that we can implement this operation in $O(|N_x|) = O(|V|)$ time. An update if the answer to this query is yes requires simply to update the 3-partition $V = S + C + Q$ of G so that it becomes a 3-partition of G_x. We are also able to implement this update operation in $O(|N_x|) = O(|V|)$ time. An update if the reply to this query is no has the choice of adding a minimal or a minimum set of additional edges to G_x to obtain a split graph. A minimal split completion of G_x by existing algorithms would require $O(|V| + |E| + |N_x|) = O(|V| + |E|)$ time, whereas we are able to implement this dynamic operation in time $O(|N_x| + |E_x|) = O(|V|)$, where E_x is the set of additional edges added to maintain the split property. In our implementation, each edge in E_x is incident to x, and thus $|N_x| + |E_x| \leq |V|$. A minimum split completion of G_x has not been studied before, and we will show that it can be done in linear time. Each of these four operations will be handled in its own subsection.

3.2.1 Query for adding a vertex

First we show under which conditions G_x is a split graph, given that G is a split graph.

Theorem 12 *Given a split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, and a vertex $x \notin V$ with a set of neighbors $N_x \subseteq V$, $G_x = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$ is a split graph if and only if one of the following holds:*

1. *$|N_x| \leq \omega(G) - 1$, $N_x \subset C \cup Q$, and Q is a clique or empty*
2. *$|N_x| \leq \omega(G) - 1$, $N_x \subset C \cup Q$, $|N_x \cap Q| \leq 1$, and Q is an independent set*
3. *$|N_x| > \omega(G) - 1$, $C \subseteq N_x$, $|N_x \cap Q| \geq |Q| - 1$, and Q is a clique*
4. *$|N_x| > \omega(G) - 1$, $C \subseteq N_x$, and Q is an independent set or empty*

Proof. Let us recall that for a split graph $G = (V, E)$ with 3-partition $V = S + C + Q$, $|C| + |Q| \geq \omega(G)$.

(Only if) For this direction of the proof, we assume that G_x is split, and we show that one of the four listed conditions must hold. Regardless of whether G_x is split or not, we have exactly the four following cases:

1. $|N_x| \leq \omega(G) - 1$, and Q is a clique or empty
2. $|N_x| \leq \omega(G) - 1$, and Q is an independent set
3. $|N_x| > \omega(G) - 1$, and Q is a clique
4. $|N_x| > \omega(G) - 1$, and Q is an independent set or empty.

Thus, for Case 1 we have to show that $N_x \subset C \cup Q$, for Case 2 we have to show that $N_x \subset C \cup Q$ and $|N_x \cap Q| \leq 1$, for Case 3 we have to show that $C \subset N_x$ and $|N_x \cap Q| \geq |Q| - 1$, and finally for Case 4 we have to show that $C \subseteq N_x$. Observe that $N_x = N_{G_x}(x)$.

For the first two cases, since $|C| + |Q| \geq \omega(G)$ and $|N_x| \leq \omega(G) - 1$, showing that $N_x \cap S = \emptyset$ gives $N_x \subset C \cup Q$.

Case 1. Assume that $|N_x| \leq \omega(G) - 1$ and Q is a clique or empty, so $|C| + |Q| = \omega(G)$. Assume for the sake of contradiction that x has a neighbor s belonging to S in G_x. Then x has at most $\omega(G) - 2$ neighbors belonging to $C \cup Q$. Observe also that, since S is non-empty, there are at least two vertices in C.

If Q is empty, $|C| = \omega(G)$, and thus there are at least two vertices c_1 and c_2 in C that are not adjacent to x in G_x (observe that, since S is non-empty, there are always at least two vertices in C). Now, if neither c_1 nor c_2 is adjacent to s, we get a $2K_2$ induced by $\{x, s, c_1, c_2\}$ contradicting the assumption that G_x is split. If both c_1 and c_2 are adjacent to s then we know that there is at least one vertex $c \notin \{c_1, c_2\}$ in C that is not adjacent to s. If $c \in N_x$ then we get a C_4 induced by $\{x, c, c_1, s\}$. If $c \notin N_x$ then we know that c must have another neighbor s_1 in S by property 7. In this case, we get a $2K_2$ induced by $\{x, s, c, s_1\}$ when s_1 is not adjacent to x, we get a C_5 induced by $\{s, x, s_1, c, c_1\}$ when s_1 is adjacent to x but not to c_1, and we get a $C_4 = \{s, x, c_1, s_1\}$ when s_1 is adjacent to both x and c_1. If only one of c_1 and c_2, say c_1, is adjacent to s, then c_2 must have a neighbor $s_1 \in S$. If $s_1 \in N_x$ then we have a $C_5 = \{x, s, c_1, c_2, s_1\}$ when c_1 is not adjacent to s_1, and we have a $C_4 = \{x, s, c_1, s_1\}$ when c_1 is adjacent to s_1. If $s_1 \notin N_x$ then we have a $2K_2 = \{x, s, c_2, s_1\}$. All of these induced subgraphs contradict the assumption that G_x is split, and thus we can conclude that $N_x \cap S$ must be empty if Q is empty.

If Q is a clique with at least two vertices, then x must have at least $|Q| - 1$ neighbors in Q or we get a $2K_2$ induced by $\{q_1, q_2, x, s\}$, for two vertices q_1 and q_2 of Q that are not adjacent to x in G_x. Since $|C| < \omega(G) - 1$ and x has at least a neighbor $q \in Q$, there exists at least one vertex $c \in C$ that is not adjacent to x in G_x. If c is adjacent to s then, we have a $C_4 = \{x, s, c, q\}$, which contradicts that G_x is split. If c is not adjacent to s then c has a neighbor s_1 in S by property 7. If s_1 is adjacent to x we get a $C_4 = \{x, s_1, c, q\}$, and if s_1 is not adjacent to x we get a $2K_2 = \{x, s, c, s_1\}$. Since all possibilities lead to an induced subgraph, contradicting that G_x is split, we can again conclude thet $N_x \cap S$ is empty when Q is a clique with at least two vertices.

If $|Q| = 1$ then $|C| = \omega(G) - 1$. Since x has a neighbor in $s \in S$, there is at least one vertex $c \in C$ that is not adjacent to x. Let q be the single vertex of Q. If x is adjacent to q we have the same possibilities that were covered above when Q was a clique with at least two vertices and x was adjacent to at least one of them. When x is not adjacent to q then we have the following cases. If c is not adjacent to s, we get a $2K_2 = \{x, s, q, c\}$ since q is adjacent to every vertex of C. If c is adjacent to s, then there is another vertex $c_1 \in C$ which is not adjacent to s, because if a vertex of S were completely connected to C, it would have degree $\omega - 1$ and it should be in Q. If c_1 is adjacent to x we get a $C_4 = \{x, s, c, c_1\}$, or we get a $2K_2 = \{x, s, c_1, q\}$. In all possible cases, we get a forbidden induced subgraph, contradicting that G_x is split when we assume that $N_x \cap S \neq \emptyset$. Thus we can conclude that $N_x \cap S = \emptyset$ and consequently $N_x \subset C \cup Q$.

Case 2. Assume that $|N_x| \leq \omega(G) - 1$, and Q is an independent set of size at least two, or we are in the previous case with $|Q| = 1$. We first show that x cannot be adjacent to more than one vertex of Q in G_x. Assume on the contrary that x is adjacent to at least two vertices q_1 and q_2 of Q. Thus there is at least one vertex c in C which x is not adjacent to, because $|C| = \omega(G) - 1$. Since every vertex of Q is adjacent to every vertex of C, the set $\{x, q_1, c, q_2\}$ induces a C_4 in G_x, contradicting that G_x is split. Hence, we have shown that $|N_x \cap Q| \leq 1$. Now, we must also show that $N_x \cap S = \emptyset$, and we assume for the sake of contradiction that x has a neighbor belonging to S in G_x, implying that x has at most $\omega(G) - 2$ neighbors belonging to $C \cup Q$. Since x has at most one neighbor in Q, we have only two possibilities: x is not adjacent to any vertices of Q, in which case it has at least one non-neighbor in C, or x is adjacent to a single vertex $q \in Q$, in which case it has at least two non-neighbors in C. The first possibility can be covered by the previous case when Q has only one element and x is not incident to it, the second possibility, when x is incident to one element of Q and Q is an independent set of size greater than 1, must be analyzed apart because property 7 does not hold in this case. When x is incident to one vertex $q \in Q$ and to a vertex $s \in S$, there exists at least a vertex $c \in C$ that is not incident to x. If c is incident to s, we get a $C_4 = \{x, s, c, q\}$, otherwise for any other vertex $q_1 \in (Q \setminus \{q\})$, we have a $2K_2 = \{x, s, c, q_1\}$. In all possible cases , we get a forbidden induced subgraph, contradicting that G_x is split. We can, then, conclude in this case too, that $N_x \cap S = \emptyset$ and consequently $N_x \subset C \cup Q$.

Case 3. Assume that $|N_x| > \omega(G) - 1$ and Q is a clique. If x has no neighbors in S, then $|N_x| = \omega(G)$, meaning that it is adjacent to all vertices of C and Q, satisfying Condition 3 of the theorem. Let us assume, then, that x has at least a neighbor $s \in S$. First we show that x has at least $|Q| - 1$ neighbors belonging to Q in G_x. Assume on the contrary that there are two vertices q_1 and q_2 in Q which are not adjacent to x in G_x. Since s is not adjacent to any vertex of Q, we get a $2K_2 = \{x, s, q_1, q_2\}$, contradicting that G_x is split. Thus x is adjacent to at least $|Q| - 1$ vertices in Q. It remains to show that all vertices of C are adjacent to x in G_x. Assume on the contrary that there is a vertex $c \in C$ which is not in N_x. Remember that by Property 7 c has at least one neighbor in S. If x has a neighbor $s_1 \in S$ which is not adjacent to c, and c has a neighbor $s_2 \in S$ which is not adjacent to x, then we get a $2K_2 = \{x, s_1, c, s_2\}$. If x and c have at least two common neighbors s_1 and s_2 in S, then we get a $C_4 = \{x, s_1, c, s_2\}$. For the last remaining case, assume that x and c have exactly one common neighbor $s_1 \in S$. If x has at least one neighbor $q \in Q$, we get a $C_4 = \{x, s_1, c, q\}$. If x has no neighbors in Q, then Q has one single element q, and x has at least two neighbors in S. At least one of these neighbors, say s_2, is different from s_1 and thus not a neighbor of c, hence we get a $2K_2 = \{x, s_2, c, q\}$. Thus if there is a vertex $c \in C$ which is not a neighbor of x in G_x, we find a forbidden subgraph in all possible cases, contradicting the assumption that G_x is split. Hence $C \subseteq N_x$.

Case 4. Assume that $|N_x| > \omega(G) - 1$, and Q is an independent set or empty. We must show that every vertex of C belongs to N_x. Assume on the contrary that there is a vertex c in C which is not adjacent to x in G_x.

If Q is empty then $|C| = \omega(G)$, and x has at least one neighbor in S, since it misses a vertex of C as neighbor. Since each vertex in S has degree at most $\omega(G) - 1$, each vertex in S misses a vertex in C as neighbor. If x has a neighbor in S which is not adjacent to c and c has a neighbor in S which is not adjacent to x, or if c and x have at least two common neighbors in S, then we get situations that are covered by the proof of Case 3. Assume that x and c have exactly one common neighbor s in S. Since s cannot be adjacent to all C, there exists at least a vertex $c_1 \in C$ that is not adjacent to s. If x is adjacent to c_1 we get a $C_4 = \{x, s, c, c_1\}$. Otherwise we can find a vertex $s_1 \in S$ adjacent to c_1 by Property 7, such that, either x is incident to s_1 and we get a $C_5 = \{x, s, c, c_1, s_1\}$, or, if s_1 is not adjacent to x, we get a $2K_2 = \{s, x, s_1, c_1\}$. Thus if Q is empty, then C must be a subset of N_x, otherwise we get a forbidden induced subgraph, contradicting that G_x is split.

If Q is an independent set of size greater than one (or we can consider it a clique), and if x has at least two neighbors q_1 and q_2 in Q, then we get a $C_4 = \{x, q_1, c, q_2\}$. Thus, since x can have at most one neighbor $q \in Q$, and there exists a vertex $c \in C$ not incident to x, x has at least one neighbor in $s \in S$. When x is incident to q, then, either c is incident to s and we get a $C_4 = \{x, s, c, q\}$, or for any other vertex $q_1 \in (Q \setminus \{q\})$, we there is $2K_2 = \{x, s, c, q_1\}$. When x has no neighbors in Q, if c is not incident

to s, we get a $2K_2 = \{x, s, c, q\}$ for any $q \in Q$, otherwise there exists a vertex $c_1 \in C$ that is not incident to s, and either we get a $2K_2 = \{x, s, c_1, q\}$ if x is not incident to c_1, or a $C_4 = \{x, s, c, c_1\}$ otherwise. All possibilities lead to a forbidden induced subgraph, contradicting that G_x is split, hence $C \subseteq N_x$ when Q is an independent of size at least two.

We have covered all possible cases, and thus the proof of this direction is complete.

(If) Now we assume that one of the four conditions listed in the theorem holds, and we show that G_x is then a split graph.

Assume that $|N_x| \leq \omega(G) - 1$, $N_x \subset C \cup Q$, and Q is a clique or empty. In this case, a split partition $V = I + K$ for G is given by $I = S$ and $K = C \cup Q$. Hence in G_x, x is adjacent only to vertices in K. Consequently, $V \cup \{x\} = I \cup \{x\} + K$ is a partition of the vertices of G_x into a clique and an independent set, and thus G_x is split.

Assume that $|N_x| \leq \omega(G) - 1$, $N_x \subset C \cup Q$, $|N_x \cap Q| \leq 1$, and Q is an independent set. In this case, a split partition $V = I + K$ for G is given by $I = S \cup (Q \setminus \{q\})$ and $K = C \cup \{q\}$, where $q \in Q \cap N_x$ if $|N_x \cap Q| = 1$, or just any vertex of Q otherwise. Then in G_x, x is adjacent only to K, so that $V \cup \{x\} = I \cup \{x\} + K$ is a split partition of G_x, and G_x is split.

Assume that $|N_x| > \omega(G) - 1$, $C \subseteq N_x$, $|N_x \cap Q| \geq |Q| - 1$, and Q is a clique. In this case, a split partition $V = I + K$ for G is given by either $I = S$ and $K = C \cup Q$ if $Q \subset N_x$, or $I = S \cup \{q\}$ and $K = C \cup (Q \setminus \{q\})$ if $Q \setminus N_x = \{q\}$. In G_x, x is adjacent to all vertices of K and possibly some vertices of I, so that $V \cup \{x\} = I + K \cup \{x\}$ is a split partition of G_x.

Assume that $|N_x| > \omega(G) - 1$, $C \subseteq N_x$, and Q is an independent set or empty. In this case, a split partition $V = I + K$ for G is given by $I = S \cup Q$ and $K = C$, so x is incident to all vertices of K and possibly some vertices of I in G_x. Then $V \cup \{x\} = I + K \cup \{x\}$ is a split partition of G_x. ∎

Theorem 13 *Given a split graph $G = (V, E)$ together with its 3-partition, a vertex $x \notin V$, and a set of neighbors $N_x \subseteq V$ of x in G, it can be decided in $O(|N_x|)$ time whether $G_x = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$ is a split graph.*

Proof. We only need to check that one of the 4 premises of the Theorem 12 is satisfied. As described in our data structure, checking whether Q is a clique, an independent set, or empty, can be done in constant time. We also keep the size of $\omega(G)$ updated at all times, so finding $\omega(G)$ takes constant time for the query operation. The remaining conditions to check all require no more than $O(|N_x|)$ time each. ∎

3.2.2 Update when the answer to vertex addition query is yes

If the answer to the query of whether adding vertex x to G results in a split graph is yes, the 3-partition $V \cup \{x\} = S' + C' + Q'$ of the updated graph G_x must be computed according to the four cases of Theorem 12. This can be done as explained in detail by the following algorithm.[1]

[1] The **If** statements of the algorithms are labeled because some later proofs and discussions refer to these cases.

Algorithm: Update-vertex-addition

Input: A split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, a vertex $x \notin V$, and a set $N_x \subseteq V$.
Output: A split graph $G' = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$, and its 3-partition $V \cup \{x\} = S' + C' + Q'$

$\omega = \omega(G)$;
$d(x) = |N_x|$;
If $d(x) \leq \omega - 1$ **and** $N_x \subset C \cup Q$ **and** Q is a clique or empty **then**
  If $d(x) = \omega - 1$ **and** $|Q \setminus N_x| = 1$ **then** *(1a)*
    Let q be the single vertex in $Q \setminus N_x$;
    $S' = S$; $C' = C \cup (N_x \cap Q)$; $Q' = \{x, q\}$;
  Else *(1b)*
    $S' = S \cup \{x\}$; $C' = C \cup (N_x \cap Q)$; $Q' = Q \setminus N_x$;
  EndIf
ElseIf $d(x) \leq \omega - 1$ **and** $N_x \subset C \cup Q$ **and** Q is an independent set **then**
  If $|N_x \cap Q| = 1$ **then** *(2a)*
    Let q be the single vertex in $N_x \cap Q$;
    $S' = S \cup \{x\} \cup (Q \setminus \{q\})$; $C' = C \cup \{q\}$; $Q' = \emptyset$;
  ElseIf $N_x \cap Q = \emptyset$ **and** $d(x) < \omega - 1$ **then** *(2b)*
    $S' = S \cup \{x\}$; $C' = C$; $Q' = Q$;
  ElseIf $N_x \cap Q = \emptyset$ **and** $d(x) = \omega - 1$ **then** *(2c)*
    $S' = S$; $C' = C$; $Q' = Q \cup \{x\}$;
  EndIf
ElseIf $d(x) > \omega - 1$ **and** $C \subseteq N_x$ **and** $|N_x \cap Q| \geq |Q| - 1$ **and** Q is a clique, **then**
  If $|N_x \cap Q| = |Q| - 1$ **then** *(3a)*
    $S' = S \cup (Q \setminus N_x)$; $C' = C \cup (N_x \cap Q) \cup \{x\}$; $Q' = \emptyset$;
  ElseIf $|N_x \cap Q| = |Q|$ **and** $N_x \cap S = \emptyset$ **then** *(3b)*
    $S' = S$; $C' = C$; $Q' = Q \cup \{x\}$;
  ElseIf $|N_x \cap Q| = |Q|$ **and** $N_x \cap S \neq \emptyset$ **then** *(3c)*
    $S' = S$; $C' = C \cup \{x\}$; $Q' = Q$;
  EndIf
ElseIf $d(x) > \omega - 1$ **and** $C \subseteq N_x$ **and** Q is an independent set or empty **then**
  If $(|N_x \cap Q| = 1$ **or** $Q = \emptyset)$ **and** $N_x \cap S = \emptyset$ **then** *(4a)*
    $S' = S \cup (Q \setminus N_x)$; $C' = C$; $Q' = (N_x \cap Q) \cup \{x\}$;
  ElseIf $|N_x \cap Q| > 1$ **or** $N_x \cap S \neq \emptyset$ **then** *(4b)*
    $S' = S \cup (Q \setminus N_x)$; $C' = C \cup \{x\}$; $Q' = N_x \cap Q$;
  EndIf
EndIf
$G' = (V \cup \{x\}, E \cup \{xy \mid y \in N_x\})$;

Lemma 14 *Given a split graph $G = (V, E)$ together with its 3-partition, a vertex $x \notin V$, a set $N_x \subseteq V$, and the knowledge that $G_x = \{V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$ is a split graph, the 3-partition of G_x is correctly computed by Algorithm Update-vertex-addition.*

Proof. When $d(x) \leq \omega - 1$, the size of the maximum clique cannot increase, so $\omega(G_x) = \omega(G) = \omega$. When $d(x) < \omega - 1$, $N_x \subset C \cup Q$, and Q is a clique or empty, then x must be in S', but the vertices in Q that are adjacent to x must be moved to C', since their degrees become ω. When $d(x) = \omega - 1$, $N_x \subset C \cup Q$, and Q is a clique or empty, then x must be in Q' with the only vertex of Q that is not adjacent to x and the rest of Q must move to C' for the same reason as the previous case; or if $Q \subseteq N_x$, all Q would move to C', so that the partition becomes unique and x must go in S' even if $d(x) = \omega - 1$. When $d(x) \leq \omega - 1$, $N_x \subset C \cup Q$, and Q is an independent set, then we need to check whether x is adjacent to a vertex $q \in Q$ or not. If it is, then we need to move q to C since its degree increased to ω,

but doing so means that $|C'| = \omega$, so the partition is unique and all vertices of $Q \setminus \{q\}$ must be moved to S'. If x is not adjacent to any vertex in Q, then only its degree matters. Hence if $d(x) < \omega - 1$ then x belongs to S', and otherwise it belongs to Q', while the rest remains unchanged.

When $d(x) > \omega - 1$, the size of the maximum clique can increase by at most one, so we define $\omega' = \omega(G_x)$. If $d(x) > \omega - 1$, $C \subseteq N_x$, $|N_x \cap Q| \geq |Q| - 1$, and Q is a clique, then $\omega' = \omega + 1$ only if $N_x \cap Q = Q$. In this case, if $d(x) = \omega$ then x belongs to Q', and otherwise to C', leaving the rest unchanged. If $|N_x \cap Q| = |Q| - 1$, then instead $d(x) \geq \omega = \omega'$, so x must belong to C' with all its neighbors in Q, meaning that the the partition of G_x is unique, and we must move $Q \setminus N_x$ to S'. When $d(x) > \omega - 1$, $C \subseteq N_x$, and Q is an independent set or empty, then $\omega' = \omega + 1$ when x is adjacent to at least one vertex of Q or when $Q = \emptyset$. Notice that in any case, all vertices in $N_x \cap Q$ will always have degree $\omega' - 1$ in G_x and therefore will belong to Q', while all vertices of $Q \setminus N_x$ will always go to S', either because they have degree $\omega' - 2$ or because the partition of G_x is unique. Besides, S is always a subset of S', because either the degree of the vertices in S increases no more than the size of the maximum clique does, or the partition of G_x is unique. Hence we only need to check where to put x. If x is adjacent to exactly one vertex of $q \in Q$ or all C when $Q = \emptyset$, but none of S, then its degree is $\omega = \omega' - 1$ meaning that it must belong to Q'. If x is adjacent to more than one vertex in Q, or at least to a vertex of S, even when $Q = \emptyset$, its degree is always greater than $\omega' - 1$, and therefore x must belong to C'.

■

Theorem 15 *Given a split graph $G = (V, E)$ together with its 3-partition, a vertex $x \notin V$, and a set of neighbors $N_x \subseteq V$ of x in G, if $G_x = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$ is a split graph then the 3-partition of G_x can be computed in time $O(|N_x|)$.*

Proof. As we explained in Section 3.1, it takes constant time to check whether Q is a clique, independent set, or empty, and the number of vertices in each set. To prove the theorem, we need to prove that for each case of the algorithm Update-vertex-addition, the 3-partition can be updated in $O(|N_x|)$ time, since it is clear that the conditions of each **If** statement can be checked in $O(|N_x|)$ time.

Moving a single vertex from one set to another takes constant time, as explained in Section 3.1. However, sometimes we need to move almost a whole set to another set. In particular, for a given subset $A \subset Q$, it might be necessary to move all vertices of $Q \setminus A$ to S (like in cases 2a, 3a, 4a, and 4b of the algorithm). However $A \subseteq N_x$, so we can access directly all vertices of A, and since we move vertices from Q, they have all degree $\omega(G) - 1$, meaning that the corresponding element in the degree array of S must be empty, since G could not have a unique partition. In this conditions we can use Lemma 11, claiming that in cases 2a, 3a, 4a and 4b, the update takes $O(|A|) = O(|N_x|)$.

In cases 1a, 1b, and 3a , we need to move a set of vertices from Q to C, but all vertices are in N_x, so we can perform this update in $O(|N_x|)$ time. ■

3.2.3 Minimal split completion of G_x when the answer to vertex addition query is no

When the answer to a vertex addition query is no, it is always possible to find a minimal split completion of G_x with additional edges that are all incident to x. This is because adding a universal vertex to a split graph always results in a split graph. Thus we know that by adding all missing edges incident to x in G_x we obtain a split completion.

Algorithm: Minimal-vertex-completion

Input: A split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, a vertex $x \notin V$, and a set $N_x \subset V$.
Output: A minimal split completion G' of $G_x = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$, and the 3-partition $V \cup \{x\} = S' + C' + Q'$ of G'.

If $N_x \cap S \neq \emptyset$ **then**
  If Q is a clique **and** $|N_x \cap Q| \leq |Q| - 1$ **then** *(1a)*
    Let q be a vertex of $Q \setminus N_x$;
    $F = \{xy \mid y \in (C \setminus N_x) \cup (Q \setminus N_x \setminus \{q\})\}$;
    $S' = S \cup \{q\}$; $C' = C \cup (Q \setminus \{q\}) \cup \{x\}$; $Q' = \emptyset$;
  ElseIf Q is a clique **and** $N_x \cap Q = Q$ **then** *(1b)*
    $F = \{xy \mid y \in C \setminus N_x\}$;
    $S' = S$; $C' = C \cup \{x\}$; $Q' = Q$;
  ElseIf Q is an independent set **then** *(1c)*
    $F = \{xy \mid y \in C \setminus N_x\}$;
    $S' = S \cup (Q \setminus N_x)$; $C' = C \cup \{x\}$; $Q' = N_x \cap Q$;
  EndIf
ElseIf $N_x \cap S = \emptyset$ **then** *(2)*
  $F = \{xy \mid y \in C \setminus N_x\}$;
  $S' = S \cup (Q \setminus N_x)$; $C' = C \cup \{x\}$; $Q' = N_x \cap Q$;
EndIf
$G' = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\} \cup F)$;

Lemma 16 *Given a split graph $G = (V, E)$ together with its 3-partition, a vertex $x \notin V$, and a set $N_x \subset V$, a minimal split completion G' of $G_x = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$ and the 3-partition of G' is correctly computed by Algorithm Minimal-vertex-completion.*

Proof. If G_x is not split, the algorithm adds additional edges incident to x in order to satisfy one of the four conditions of Theorem 12.

If $N_x \cap S \neq \emptyset$, we can choose only condition 3 or 4 depending on Q, because we proved that $N_x \cap S = \emptyset$ otherwise. If Q is a clique, we make x adjacent to all vertices in C and at least $|Q| - 1$ vertices in Q. If Q is an independent set, we make x adjacent to all vertices of C, producing a split completion by condition 4.

If $N_x \cap S = \emptyset$, Q must be an independent set, or G_x would be always split satisfying either condition 1 of Theorem 12 if $|N_x| \leq \omega(G) - 1$, or condition 3 if $|N_x| = \omega(G) - 1$. (The size of N_x cannot be larger since $|C| + |Q| = \omega$, so we must have $N_x \cap C = C$). Furthermore, we have $|N_x \cap Q| > 2$, or condition 2 of Theorem 12 would be satisfied, and $N_x \cap C \neq C$ or condition 4 would be satisfied instead. The only possible choice is, therefore, to add edges to satisfy condition 4. This is always possible by making x adjacent to all vertices of C.

For all cases, it is easy to check that the addition of set F to G_x makes the resulting graph obey one of the conditions of Theorem 12, and hence the produced graph G' is split. The computation of sets S', C', and Q' are completely analogous to the corresponding cases of Algorithm Update-vertex-addition, and hence these sets are correctly computed by the proof of Lemma 14. In particular the correspondence between this algorithm and Algorithm Update-vertex-addition is the following: the first case corresponds to case 3a; the second case to case 3c; the third and the fourth to case 4b. Finally, observe that in each case, both endpoints of every edge in F belong to C', ensuring that the computed split completion is minimal by Theorem 8. ∎

Theorem 17 *Given a split graph $G = (V, E)$ together with its 3-partition, a vertex $x \notin V$, and a set $N_x \subset V$, a minimal split completion G' of $G_x = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$ and the 3-partition of G' can be computed in time $O(d_{G'}(x))$.*

Proof. Computing and adding the set of fill edges F by Algorithm Minimal-vertex-completion, takes $O(|N_x| + |F|)$ time. In fact we always need to make all vertices of C adjacent to x, meaning that we have to scan the set C to find the vertices that are not neighbors with x, but $|C| \leq |N_x| + |F|$, so this operation takes $O(|N_x| + |F|)$ time. Sometimes we also need to scan Q, but then we will make x adjacent to all vertices of Q except one vertex, meaning that $|C| + |Q| - 1 = |N_x| + |F|$, so that $O(|C| + |Q|) = O(|N_x| + |F|)$. To update the 3-partition after the addition of the fill edges, can be done $O(|N_x| + |F|)$ by Theorem 15, since it is simply a vertex addition where the new vertex x has neighborhood $N_x \cup \{y \mid xy \in F\}$ in G, and the resulting graph is split. Thus the total running time is $O(|N_x| + |F|)$, and since all edges in F are incident to x, $|N_x| + |F| = d_{G'}(x)$ and the result follows. ∎

Observe that Algorithm Minimal-vertex-completion can be simplified. We can simply compute the set F, and then call Algorithm Update-vertex-addition with G, x, and $N_x \cup \{y \mid xy \in F\}$ as input, to compute G' and its 3-partition, since we know that G' is split. However for the proof of correctness and the running time, we found it useful to explicitly show how the 3-partition changes in each case.

We end this section with the following observation which we find interesting on its own, and which will be useful in the next section.

Observation 18 *Given a split graph $G = (V, E)$ and a vertex x with a set of neighbors $N_x \subseteq V$ in G, all possible minimal split completions of $G_x = (V \cup \{x\},\ E \cup \{xy \mid y \in N_x\})$, where all fill edges are incident to x, have the same number of edges.*

Proof. As can be seen in Algorithm Minimal-vertex-completion and the proof of Lemma 16, either we have to make x adjacent to all vertices of C, in which case the minimal completion is unique, or we have to make x adjacent to all vertices of C and all but one vertices of Q, in which case all minimal completions have the same number of fill edges. ∎

3.2.4 Minimum split completion of G_x when the answer to vertex addition query is no

In this section we will show that it is possible to compute, in linear time, a minimum split completion of a split graph plus one vertex, when adding such vertex does not keep the graph split.

Theorem 19 *Given a split graph $G = (V, E)$ with 3-partition $V = S + C + Q$, a vertex $x \notin V$, a set $N_x \subseteq V$, and the knowledge that $G_x = (V,\ E \cup \{xy \mid y \in N_x\})$ is not split, a minimum set of fill edges that add to G_x to make it split again, is the smallest between the following two possible sets of fill edges F_1 and F_2:*

1. *If $N_x \cap S \neq \emptyset$ and Q is a clique:*

   - *Define $Q_F = \emptyset$ if $N_x \cap Q = Q$, or $Q_F = Q \setminus N_x \setminus \{q\}$ with $q \in Q \setminus N_x$ otherwise, then:*
     $F_1 = \{xv \mid v \in (C \setminus N_x) \cup Q_F\}$ and
     $|F_1| = |C \setminus N_x| + |Q_F|$
   - *Define $Q_F = Q$ if $N_x \cap Q = Q$, or $Q_F = Q \setminus \{q\}$ with $q \in Q \setminus N_x$ otherwise, then:*
     $F_2 = \{sv \mid s \in N_x \cap S \wedge v \in (C \cup Q_F \cup (N_x \cap S)) \wedge sv \notin E\}$
     $|F_2| = |N_x \cap S| \cdot \max\{|Q| - 1, |N_x \cap Q|\} + \sum_{s \in N_x \cap S} |C \setminus N_G(s)| + |N_x \cap S| \cdot (|N_x \cap S| - 1)/2$

2. *$N_x \cap S \neq \emptyset$ and Q is an independent set:*

   - *$F_1 = \{xv \mid v \in C \wedge xv \notin E\}$ and*
     $|F_1| = |C \setminus N_x|$
   - *$F_2 = \{sv \mid s \in N_x \cap (S \cup Q) \wedge v \in C \cup (N_x \cap (S \cup Q)) \wedge sv \notin E\}$*
     $|F_2| = |N_x \cap (S \cup Q)| \cdot (|N_x \cap (S \cup Q)| - 1)/2 + \sum_{s \in N_x \cap S} |C \setminus N_G(s)|$

3. *$N_x \cap S \neq \emptyset$ and $Q = \emptyset$:*

- $F_1 = \{xv \mid v \in C \wedge xv \notin E\}$ *and*
  $|F_1| = |C \setminus N_x|$
- $F_2 = \{sv \mid s \in N_x \cap S \wedge v \in C \cup (N_x \cap S) \wedge sv \notin E\}$
  $|F_2| = |N_x \cap S| \cdot (|N_x \cap S| - 1)/2 + \sum_{s \in N_x \cap S} |C \setminus N_G(s)|$

4. $N_x \cap S = \emptyset$*:*

   - $F_1 = \{xv \mid v \in C \wedge xv \notin E\}$ *and*
     $|F_1| = |C \setminus N_x|$
   - $F_2 = \{qv \mid q, v \in N_x \cap Q \wedge qv \notin E\}$
     $|F_2| = |N_x \cap Q| \cdot (|N_x \cap Q| - 1)/2$

Proof. Given a split graph $G = (V, E)$ and a new vertex $x \notin V$ with a given neighborhood $N_x \subseteq V$, such that $G_x = (V, E \cup \{xy \mid y \in N_x\})$ is not split, we have only two options to minimally complete G_x into a split graph. Either we add a minimal set of fill edges $F_1 = F$ defined by Algorithm Minimal-vertex-completion, where all edges are incident to the new vertex x; or we add a minimal set of fill edges $F_2 = F_G \cup F_x$, where all fill edges in F_G have both endpoints in V, $G' = (V, E \cup F_G)$ is split, and all fill edges in $F_x \neq F_1$ are incident to x. F_x might be empty, but we only need to consider sets F_G that are not empty, since otherwise by Observation 18 $|F_x| = |F_1|$. If we want to find a minimum completion, we need to take the smallest between F_1 (we know by Observation 18 that all such sets F_1 have the same cardinality) and all possible sets F_2. Observe that if $F_1 \subset F_2$, then F_2 does not yield a minimal completion, and if $|F_2| \geq |F_1|$, we can ignore F_2.

If we are given a set of fill edges $F_2 = F_G \cup F_x$, $H = (V \cup \{x\}, E \cup F_2)$. If we are given a set of fill edges F_1, then we define $H = (V \cup \{x\}, E \cup F_1)$. As far as the notation is concerned, we use $V = S + C + Q$ for the 3-partition of the input split graph $G = (V, E)$, we use $V = S' + C' + Q'$ for the 3-partition of the graph G', and we use $V_H = S_H + C_H + Q_H$ for the 3-partition of the split completion H.

We will now show that, in each possible case, the two set of fill edges given by the theorem, are the only two possible minimal completions we can choose between to find the minimum.

1. $N_x \cap S \neq \emptyset$ and Q is a clique. Then, either case 1a or 1b of Algorithm Minimal-vertex-completion applies, so that x becomes adjacent to all vertices of C and at least $|Q| - 1$ vertices in Q; or we modify the 3-partition into $V = C' + S' + Q'$ in some other way, and make the new graph satisfy one of the conditions of Theorem 12. Since we assume that G_x is not split, we know that in G_x, x misses either at least one vertex of C as a neighbor or at least two vertices of Q as neighbors.

   - If we add fill edges incident to x, we should decide which of cases 1a and 1b of Algorithm Minimal-vertex-completion should apply, checking whether $N_x \cap Q = Q$. If we have this equality, we define $Q_F = \emptyset$, otherwise $Q_F = Q \setminus N_x \setminus \{q\}$, where we can choose q as any vertex in $Q \setminus N_x$. Therefore we get:

     $$F_1^1 = \{xv \mid v \in (C \setminus N_x) \cup Q_F\}$$
     $$|F_1^1| = |C \setminus N_x| + |Q_F|$$

   - If we want to add a set of edges F_G to G and try to satisfy condition 1 or 2 of Theorem 12, we have to create a new partition $V = C' + S' + Q'$ such that $N_x \cap S' = \emptyset$. To do this, we need to bring all vertices of $N_x \cap S$ into $C' \cup Q'$. This means that every vertex in $N_x \cap S$ must become adjacent to at least all vertices of C and $|Q| - 1$ vertices of Q, since their degree must increase at least to $\omega(G) - 1 = |C| + |Q| - 1$. We simply make x incident to all vertices of C and to the same $|Q| - 1$ vertices in Q. (For counting the fill edges at this step, this distinction is not important, but actually it can be shown that if we do not make x incident to the same vertices of Q, we might risk getting a non-minimum completion.) At this point we have $C' = C \cup Q \setminus \{q\}$, $Q' = (N_x \cap S) \cup \{q\}$ and $S' = S \setminus N_x$, where q is the vertex of Q which

we have not added fill edges incident to. We now need to distinguish between the following cases.

If $|N_x \cap S| = 1$ and we can choose q such that $q \notin N_x$ (in other words $N_x \cap Q \neq Q$), the current graph G' when added the new vertex x with neighborhood N_x, would satisfy case 2 of theorem 12. So defining $F_2^1 = F_G = \{sv \mid s \in N_x \cap S \wedge v \in (C \cup Q \setminus \{q\}) \wedge sv \notin E\}$, we get a split completion H of G_x. Furthermore, applying case 2a of Algorithm Update-vertex-addition to G' and x, we can check that in the 3-partition of the output graph H, all fill edges are in C_H, meaning that the completion is minimal.

If $|N_x \cap S| > 1$ or $Q \cap N_x = Q$, adding only F_G as defined before, we would always have more than a vertex in Q' incident to x, and since Q' is an independent set, neither condition 2 nor 4 of Theorem 12 would be satisfied, in fact there is always a vertex non-adjacent to x in C'. Adding edges between x and C' would satisfy condition 4, while we want to satisfy conditions 1 or 2. Hence we can only make $N_x \cap Q'$ into a clique to satisfy condition 1, since trying to satisfy condition 2, would require to satisfy first condition 1, and then add more edges to make Q' into an independent set. Making $N_x \cap Q'$ into a clique would make G'_x satisfy condition 1 of theorem 12, without adding any edge incident to x. So we can define $F_2^1 = F_G = \{sv \mid s \in N_x \cap S \wedge v \in (C \cup Q_F \cup (N_x \cap S))\}$, where $Q_F = Q \setminus \{q\}$ if $Q \cap N_x \neq Q$, and $Q_F = Q$ otherwise. Running Algorithm Update-vertex-addition on G' and x would output a graph H with 3-partition corresponding to case 1b of the algorithm (we have $N_x \cap Q' = Q$ in any case), proving that all fill edges are in C_H and therefore the completion is minimal.

We can summarize all possible cases as follows:

$$F_2^1 = \{sv \mid s \in N_x \cap S \wedge v \in (C \cup Q_F \cup (N_x \cap S)) \wedge sv \notin E\}$$

$$|F_2^1| = |N_x \cap S| \cdot \max\{|Q| - 1, |N_x \cap Q|\} \; + \sum_{s \in N_x \cap S} |C \setminus N_G(s)| \; + \; |N_x \cap S| \cdot (|N_x \cap S| - 1)/2$$

- If we want to satisfy conditions 3 or 4 of Theorem 12, adding a non-empty set of edges F_G to G and possibly a set of edges F_x incident to x, observe that a necessary condition is $N_x \cap C' = C'$ in any case, and $|N_x \cap Q'| \geq |Q'| - 1$ if Q' is a clique. We will define the set of fill edges in this case as $F_3^1 = F_G + F_x$.

  Now, no matter how we change the 3-partition by adding a set of edges F_G to G, since $C \cup Q$ is a clique, we know the following about G': if Q' is an independent set or empty, then $|C' \cap (C \cup Q)| \geq |C \cup Q| - 1$, and if Q' is a clique, then $|(C' \cup Q') \cap (C \cup Q)| \geq |C \cup Q| - 1$. In the first case, $|(C \cup Q) \cap N_x| - 1 \geq |F_1^1| - 1$ vertices non-adjacent to x are still in C', and since we must make x adjacent to all vertices of C', we get $|F_3^1| = |F_x| + |F_G| \geq |F_1^1| - 1 + |F_G|$. Hence we can never have $|F_3^1| < |F_1^1|$, because $|F_G| \geq 1$. In the second case, $|F_x| \geq |F_1^1| - 2$, because at least $|(C \cup Q) \cap N_x| - 1 \geq |F_1^1| - 1$ vertices non-adjacent to x are still in $C' \cup Q'$, but we might also be able to choose a vertex of $Q' \cap (C \cup Q)$ not to be adjacent to x in G', that had to be adjacent to x in G, meaning that that $|F_3^1| = |F_x| + |F_G| \geq |F_1^1| - 2 + |F_G|$. In this situation, a necessary condition to get $|F_3^1| < |F_1^1|$ is that $|F_G| = 1$ and $|(C' \cup Q') \cap (C \cup Q)| = |C \cup Q| - 1$. However, to achieve this we must have $\omega(G') \geq \omega(G) - 1$, adding just one edge to the graph, since a vertex that was in $C \cup Q$ is now in S'. A necessary condition do this, is adding an edge between two vertices of degree at least $\omega(G) - 1$. This is impossible because we assumed that Q is a non-empty clique, therefore all vertices of degree at least $\omega(G) - 1$ are in $C \cup Q$ and they are completely connected.

  We conclude that it is impossible to get $|F_3^1| < |F_1^1|$, trying to satisfy condition 3 or 4 of Theorem 12.

2. $N_x \cap S \neq \emptyset$ and Q is an independent set. In this case, we can either make x adjacent to all vertices of C, according to case 1c of Algorithm Minimal-vertex-completion; or we try to satisfy the conditions of Theorem 12 in some other way, adding fill edges also between vertices of G.

- Applying case 1c of Algorithm Minimal-vertex-completion, we get:

$$F_1^2 = \{xv \mid v \in C \wedge xv \notin E\}$$
$$|F_1^2| = |C \setminus N_x|$$

- To satisfy condition 1 or 2 of Theorem 12, as in the previous case, we need at least to move all neighbors of x in S to $C' \cup Q'$ since a necessary condition is that we create a new partition $V = C' + S' + Q'$ with $N_x \cap S' = \emptyset$. Since in this case Q is an independent set, it is enough to make every vertex in $N_x \cap S$ incident to all vertices of C, so that $Q' = (N_x \cap S) \cup Q$.

  If $|N_x \cap (Q \cup S)| = 1$, we are actually done, because G' and x would satisfy condition 2 of Theorem 12. But in the opposite case, since $N_x \cap C' \neq C'$, our only option is to make $N_x \cap (Q \cup S)$ into a clique to satisfy, instead, condition 1. In fact, making x adjacent to all vertices in C', would satisfy condition 4, while we want to satisfy either condition 1 or 2, and trying to satisfy condition 2, would require to satisfy first condition 1, and then add more edges to make Q' into an independent set.

  In both cases the completion is minimal, in fact all fill edges are in C_H, where H is the minimal completion given as output of cases 1b and 2a of the algorithm Update-vertex-completion, run with input G' and x.

  We can write the general set of fill edges as follows:

$$F_2^2 = \{sv \mid s \in N_x \cap (S \cup Q) \wedge v \in C \cup (N_x \cap (S \cup Q)) \wedge sv \notin E\}$$
$$|F_2^2| = |N_x \cap (S \cup Q)| \cdot (|N_x \cap (S \cup Q)| - 1)/2 + \sum_{s \in N_x \cap S} |C \setminus N_G(s)|$$

- If we want to satisfy cases 3 or 4 of Theorem 12, adding a non empty set of edges F_G to G and possibly a set of edges F_x incident to x, we have to remember that a necessary condition is that $N_x \cap C' = C'$ in any case, and $|N_x \cap Q'| \geq |Q'| - 1$ if Q' is a clique.

  No matter how we add edges to modify the graph, we have that if Q' is an independent set or empty, then $|C' \cap C| \geq |C| - 1$, and if Q' is a clique then $|C \cap (C' \cup Q')| \geq |C| - 1$.

  In the first case, $|C \cap N_x| - 1 \geq |F_1^2| - 1$ vertices non-adjacent to x are still in C', and since we must make x adjacent to all vertices of C', we get $|F_3^2| = |F_x| + |F_G| \geq |F_1^2| - 1 + |F_G|$. Hence it can never be that $|F_3^2| < |F_1^2|$, since $|F_G| \geq 1$.

  In the second case, $|F_x| \geq |F_1^2| - 2$, because at least $|C \cap N_x| - 1 \geq |F_1^2| - 1$ vertices non-adjacent to x are still in $C' \cup Q'$, but we might also be able to choose a vertex of $Q' \cap C$ not to be adjacent to x in G', that had to be adjacent to x in G, meaning that $|F_3^2| = |F_x| + |F_G| \geq |F_1^2| - 2 + |F_G|$. However, if a vertex that was in C is now not in $C' \cup Q'$, it must be in S'. For this to happen, the size of the maximum clique of G' must be at least $\omega(G') \geq \omega(G) + 2$ (it could be $\omega(G') \geq \omega(G) + 1$ if G' had a unique partition, but we assumed that $Q' \neq \emptyset$). Since it is not possible to increase the size of the maximum clique of G by 2 adding fewer than two edges, then $|F_G| \geq 2$ and it cannot be $|F_3^2| < |F_1^2|$.

  We conclude that it is impossible to get $|F_3^2| < |F_1^2|$ if we want to satisfy condition 3 or 4 of Theorem 12.

3. $N_x \cap S \neq \emptyset$ and $Q = \emptyset$. In this case we can apply the previous cases with $N_x \cap Q = \emptyset$ for F_2, which means:

$$F_2^3 = \{sv \mid s \in N_x \cap S \wedge v \in C \cup (N_x \cap S) \wedge sv \notin E\}$$
$$|F_2^3| = |N_x \cap S| \cdot (|N_x \cap S| - 1)/2 + \sum_{s \in N_x \cap S} |C \setminus N_G(s)|$$

4. $N_x \cap S = \emptyset$. In this case, the only way the new graph can fail to be split is if Q is an independent set and $|N_x \cap Q| > 1$ with $N_x \cap C \neq C$. What we can do then is to either make x adjacent to C, or make $N_x \cap Q$ into a clique.

   - Using case 2a of Algorithm Minimal-vertex-completion we get:

$$F_1^4 = \{xv \mid v \in C \wedge xv \notin E\}$$
$$|F_1^4| = |C \setminus N_x|$$

   - To satisfy condition 1 or 2 of Theorem 12, we can just use the same argument as for F_2^2 in the previous point, considering the special case $N_x \cap S = \emptyset$, so that we just make $N_x \cap Q$ into a clique, then:

$$F_2^4 = \{qv \mid q, v \in N_x \cap Q \wedge qv \notin E\}$$
$$|F_2^4| = |N_x \cap Q| \cdot (|N_x \cap Q| - 1)/2$$

   - The argument for showing that it is not possible to satisfy conditions 3 and 4 with fewer than $|F_1^4|$ edges, is the same as for F_3^2.

■

Theorem 20 *Given a split graph $G = (V, E)$, a vertex $x \notin V$, and a set $N_x \subseteq V$, a minimum split completion H of $G_x = (V,\ E \cup \{xy \mid y \in N_x\})$ can be computed in $O(|V| + |E|)$ time.*

Proof. As far as the running time is concerned, given the previous theorem, we have to compute the following things: The 3-partition of G; The minimum set of fill edges; The graph H. We will show that each of them can be computed in time at most $O(|V| + |E|)$.

The 3-partition of G, that can be computed in $O(|V| + |E|)$ time as described in Section 3.1. Then, it takes time $O(|N_x|) = O(|V|)$ to find out which is the smallest set between F_1 and F_2, chosen according to Theorem 19. In fact, to evaluate the formulas, we only need to know the size of the sets of the 3-partition of G, that are given, to compute the size of their intersections with N_x, that can be done in $O(|N_x|) = O(|V|)$ time, and compute the value $|C \setminus N_G(s)|$ for every $s \in N_x \cap S$, that can be done in constant time for each s since it is equivalent to compute $|C| - d_G(s)$. Besides $|F_{min}| \leq |F_1| \leq |V|$, so we will never update more than $O(|V|)$ vertices when we add F_{min} to G.

Once we know which set of fill edges to use, we need to find the endpoints of the edges that we want to add, so that we can update their degrees and adjacency lists. Computing F_1 takes $O(d_G(x) + |F_1|) = O(|V|)$ time as we have shown in the proof of theorem 17. When we need to compute F_2, we can find the fill edges to be added between vertices of $N_x \cap (S \cup Q)$ when Q is an independent set, or between $N_x \cap S$ and Q, when Q is a clique, in $O(|F_2|) = O(|V|)$ time. We know, in fact, that such vertices are all disconnected, so we need to add edges to each vertex we scan. However, to find all non-neighbors in C for every vertex $s \in N_x \cap S$, takes time $O(|C||N_x \cap S|)$, that can be much higher then the actual number of fill edges we will add, namely $\sum_{s \in N_x \cap S} |C \setminus N_G(s)|$. On the other hand, we have that $|C||N_x \cap S| \leq \sum_{s \in N_x \cap S} d_H(s) = O(|N(H)| + |E(H)|)$, in fact in H, every vertex that was in $N_x \cap S$ has been made adjacent to at least all C. We can then claim that, finding F_2, takes at most time $O(|N(H)| + |E(H)|) = O(|V| + 1 + |E| + |F_2|) = O(|V| + |E|)$ since $|F_2| = |F_{min}| = O(|V|)$.

Therefore we can always compute H in $O(|V| + |E|)$ time. ■

3.3 Adding an edge between two existing vertices in G

In this section, we describe the query and update operations for adding a new edge ab between two non-adjacent vertices a and b of a given split graph $G = (V, E)$. A query for adding this edge is simply deciding whether $G_{ab} = (V, E \cup \{ab\})$ is a split graph. An update if the answer to this query is yes requires simply to update the 3-partition $V = S + C + Q$ of G so that it becomes a 3-partition of G_{ab}. We can perform both a query and an update in $O(1)$ time, matching the running time given in [12] for the same operations.

In the case the query replies that G_{ab} is not split, then the update operation has the choice of adding a minimal or a minimum set of additional edges to G_{ab} to obtain a split graph. Computing a minimal split completion G' of G_{ab} by existing algorithms would require $O(|V| + |E|)$ time, whereas we are able to implement this dynamic operation in time $O(\max\{d_{G'}(a), d_{G'}(b)\})$, which is never worse than $O(|V|)$. A minimum split completion of G_{ab} has not been studied before, and we show that it can be done in the same time as the minimal completion. Each of these four operations will be handled in its own subsection.

3.3.1 Query for adding an edge

Lemma 21 *Given a split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, and two vertices $a, b \in V$ such that $ab \notin E$, $G_{ab} = (V, E \cup \{ab\})$ fails to be a split graph if and only if $a, b \in S$.*

Proof. There are four possibilities regarding which set each of a and b belongs to: they can both belong to S, one can belong to S and the other to either C or Q, or they can both belong to Q (only if Q is an independent set). We do not have the possibility that they both belong to C since C is a clique in G, and we do not have the possibility that one belongs to Q and the other to C since all possible edges between C and Q are already present in G. We will show that G_{ab} is not split exactly in the first possibility.

Assume that $a, b \in S$. By the definition 3-partition and Lemma 3, every vertex in S has at most $|C| - 1$ neighbors in C if $|C| \geq \omega(G) - 1$, and at most $|C|$ neighbors if $|C| < \omega(G) - 1$. Therefore we can distinguish between these two situations. Let us first consider $|C| < \omega(G) - 1$. Then there exist, by Lemma 6, $x, y \in Q$ such that $G_{ab}[a, b, x, y] = 2K_2$, thus G_{ab} is not split. Let us now consider $|C| \geq \omega(G) - 1$. If $N(a) \subseteq N(b)$ and $|N(b)| < |C| - 1$, then there exist $x, y \in C \setminus N(b)$ such that $G_{ab}[a, b, x, y] = 2K_2$. If $N(a) \subseteq N(b)$ and $|N(b)| = |C| - 1$, then there exists $x \in C \setminus N(b)$. In addition, if $|C| = \omega(G)$ x has a neighbor $y \in S$ by Property 7, otherwise Q is not empty and x has a neighbor $y \in Q$. Hence $G_{ab}[a, b, x, y] = 2K_2$. The case when $N(a) \subseteq N(b)$ is symmetric. If $N(a)$ is not a subset of $N(b)$ and $N(b)$ is not a subset of $N(a)$ then there exist $x \in C \setminus N(a)$ and $y \in C \setminus N(b)$, such that $G_{ab}[a, b, x, y] = C_4$. Since all possibilities lead to a forbidden induced subgraph, G_{ab} is not a split graph when both a and b belong to S.

Assume that $a \in S$ and $b \in C$. In this case every possible split partition of G is also a split partition of G_{ab}, because ab is always between the independent set and the clique of any partition. Thus G_{ab} is split.

Assume that $a \in S$ and $b \in Q$. We can always choose a split partition of G where b is in the clique, since it is always possible to put at least one vertex of Q in the clique. This will be a split partition also for G_{ab} since a is always in the independent set in any split partition. Thus G_{ab} is split.

Assume that $a, b \in Q$. Since Q is an independent set, $V = I + K$ is a split partition of G, where $I = S \cup (Q \setminus \{a\})$ and $K = C \cup \{a\}$. Thus, since edge ab connects I to K, $V = I + K$ is also a split partition for G_{ab}. ■

Thus to answer the query for edge addition, we only need to check whether both a and b belong to S, and the time bound follows.

Theorem 22 *Given a split graph $G = (V, E)$, its 3-partition, and two vertices $a, b \in V$ such that $ab \notin E$, it can be decided in $O(1)$ time whether $G_{ab} = (V, E \cup \{ab\})$ is a split graph.*

3.3.2 Update when the answer to edge addition query is yes

Given a split graph $G = (V, E)$ together with its 3-partition $V = C + S + Q$, and a pair of vertices $a, b \in V$ such that $ab \notin E$, if $G_{ab} = (V, E \cup \{ab\})$ is a split graph, in this section, we will show how to compute the 3-partition $V = C' + S' + Q'$ of G_{ab}.

Algorithm: Update-edge-addition

Input: A split graph $G = (V, E)$ with 3-partition $V = S + C + Q$, and a pair of vertices $a, b \in V$ such that $ab \notin E$.

Output: A split graph $G_{ab} = (V, E \cup \{ab\})$ and its 3-partition $V = S' + C' + Q'$.

$d(b) = |N_G(b)|$;
$\omega = \omega(G)$;
If $a \in C$ and $b \in S$ **then**:
  If If $d(b) = \omega - 1$ **then** *(1a)*
    $W = \{v \in C \mid d(v) = \omega\}$;
    $C' = C \setminus W$; $S' = S \setminus \{b\}$; $Q' = Q \cup W \cup \{b\}$;
  ElseIf $d(b) = \omega - 2$ **and** $Q \neq \emptyset$ **then** *(1b)*
    $C' = C$; $S' = S \setminus \{b\}$; $Q' = Q \cup \{b\}$;
  ElseIf $d(b) \leq \omega - 3$ **or** $(d(b) = \omega - 2$ **and** $Q = \emptyset)$ **then** *(1c)*
    $C' = C$; $S' = S$; $Q' = Q$;
  EndIf
ElseIf $a \in Q$ **and** $b \in Q$ **then**
  If $|Q| = 2$ **then** *(2a)*
    $W = \{v \in C \mid d(v) = \omega\}$;
    $C' = C \setminus W$; $S' = S$; $Q' = Q \cup W$;
  ElseIf $|Q| > 2$ **then** *(2b)*
    $C' = C$; $S' = S \cup Q \setminus \{a, b\}$; $Q' = \{a, b\}$;
  EndIf
ElseIf $a \in Q$ and $b \in S$ **then**
  If Q is an independent set **then** *(3a)*
    $C' = C \cup \{a\}$; $S' = S \cup Q \setminus \{a\}$; $Q' = \emptyset$;
  ElseIf Q is a clique of size greater than 1, **then**
    If $d(b) = \omega - 2$ **then** *(3b1)*
      $C' = C \cup \{a\}$; $S' = S \setminus \{b\}$; $Q' = Q \cup \{b\} \setminus \{a\}$;
    ElseIf $d(b) < \omega - 2$ **then** *(3b2)*
      $C' = C \cup \{a\}$; $S' = S$; $Q' = Q \setminus \{a\}$;
    EndIf
  EndIf
EndIf
$G_{ab} = (V, E \cup \{ab\})$;

Lemma 23 *Given a split graph $G = (V, E)$ together with its 3-partition, a pair of vertices $a, b \in V$ such that $ab \notin E$, and the knowledge that $G_{ab} = (V, E \cup \{ab\})$ is split, Algorithm Update-edge-addition correctly computes the 3-partition of G_{ab}.*

Proof. To prove correctness, we show that all possible cases are covered by the algorithm, and that the update is indeed correct for each case. We have three main possibilities: $a \in C, b \in S$ or $a, b \in Q$ or $a \in Q, b \in S$ (the cases $a \in S, b \in C$ and $a \in S, b \in Q$ are symmetric to the first and last possibility listed). For any vertex b of S, we know that $d(b) \leq \omega - 1$. Furthermore, when Q is a clique of size at least two, then we know that $d(b) \leq \omega - 2$ for each $b \in S$, since the largest clique contains at least two vertices of Q, and no vertex of S is adjacent to any vertex of Q. Thus the cases studied by the algorithm cover all possible cases that can occur.

Assume $a \in C$ and $b \in S$. If $d(b) = \omega - 1$ and $b \in S$, it means that the split partition of G is unique. When we add edge ab, the size of the maximum clique increases by one, which means that $\omega = \omega(G_{ab}) - 1$ and the partition is not unique any longer, so b and its neighbors in C with degree ω must be moved in Q'. (These vertices are in fact those vertices of C whose only neighbor outside of C was b before the addition of edge ab). If $d(b) = \omega - 2$ and $Q \neq \emptyset$, then adding an edge incident to b makes its degree increase to $\omega - 1$, and since $\omega = \omega(G_{ab})$, it should be moved to Q'. This is possible because if the degree of b became $\omega - 1$, it means that $|C| = \omega - 1$ and therefore Q is an independent set by Lemma 6. The remaining possibilities are that either $d(b) \leq \omega - 3$ or it is $\omega - 2$ and $Q = \emptyset$. In either case, the size of the maximum clique does not increase, and the degree of b does not increase enough to be moved out of S, so the 3-partition does not change.

Assume both a and b belong to Q. If $|Q| = 2$, then when we add edge ab, the size of the maximum clique increases by one, so $\omega = \omega(G_{ab}) - 1$. Therefore, if there is some vertex in C with degree $\omega = \omega(G_{ab}) - 1$, it must be moved to Q'. If $|Q| > 2$, this guarantees that all vertices in C have degree strictly greater than ω. Therefore, even if the size of the maximum clique increases by one, these vertices remain in C'. On the other hand, all other vertices in $Q\{a, b\}$ get degree $\omega(G_{ab}) - 2 = \omega - 1$, so they must be moved to S'.

Assume that $a \in Q$ and $b \in S$. If Q is an independent set, then when we add edge ab, the size of the maximum clique does not increase, so $\omega = \omega(G_{ab})$, but the degree of a increases to ω, so we have to move a to C'. After this, $|C'| = \omega$ meaning that the partition is unique by Lemma 3. Hence we must move all other vertices in $Q \setminus \{a\}$ to S'. Assume for the rest of the proof that Q is a clique of size at least 2. If $d(b) = \omega - 2$, since $|Q| \geq 2$, then $|C| = \omega - 2$. Adding the new edge does not increase the maximum clique size, so $\omega = \omega(G_{ab})$, and the degree of b becomes $\omega - 1$, while the degree of a becomes ω. Hence we must move b to Q', and a to C'. The case when $d(b) < \omega - 2$ is like the previous case, but now the degree of b cannot get high enough for it to be moved to Q', so we only need to move a to C'. ∎

Theorem 24 *Given a split graph $G = (V, E)$ together with its 3-partition, a pair of vertices $a, b \in V$ such that $ab \notin E$, and the knowledge that $G_{ab} = (V, E \cup \{ab\})$ is split, the 3-partition of G_{ab} can be computed in time $O(1)$.*

Proof. Proving this theorem is equivalent to proving that in each case of the algorithm Update-edge-addition, we can update the 3-partition in time $O(1)$, since it is clear that it takes time $O(1)$ to check an **if** statement.

For all cases, to move only a or b or both, can be done in constant time.

In cases 1a and 2a of the algorithm we might have to move up to $O(n)$ vertices from C to Q, and in 2b and 3a we might have to move up $O(n)$ vertices from Q to S.

However, for the first two cases, simply observe that the vertices that must be moved from C are exactly all vertices of C with degree $\omega(G)$, meaning that we move a whole degree list of C to Q, and Q is empty or contains only 2 vertices. In this case we can apply Lemma 11 in the special case that $A = \emptyset$, meaning that these operations can be performed in time $O(1)$.

In the latter two cases, 2b and 3a, we can apply Lemma 11 defining $A = \{a, b\}$ or $A = \{a\}$, respectively, and $X_d = Q \setminus A$. In fact the elements of Q we move into S have degree $\omega(G) - 1$, and there cannot be vertices of the same degree in S, since G does not have a unique partition. It follows that we can perform these operations in time $O(|A|) = O(1)$.

We have thus shown that the 3-partition of G_{ab} can be always computed in time $O(1)$. ∎

3.3.3 Minimal split completion of G_{ab} when the answer to edge addition query is no

Given a split graph $G = (V, E)$, and a non-edge ab, if $G_{ab} = (V, E \cup \{ab\})$ is not split, then by Theorem 21, this means that $a, b \in S$. In this case it is possible to add a set of fill edges F incident to either a or b to get a minimal split completion G' of G_{ab}. Let us choose a, noting that everything still holds if we choose b.

Algorithm: Minimal-edge-completion

Input: A split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, and a pair of vertices $a, b \in V$ such that $ab \notin E$.

Output: A minimal split completion G' of $G_{ab} = (V, E \cup \{ab\})$ and the 3-partition $V = S' + C' + Q'$ of G'.

$N(a) = N_G(a)$;
$\omega = \omega(G)$;
If Q is an independent set **then** *(1)*
  $F = \{av \mid v \in C \setminus N(a)\}$;
  $C' = C \cup \{a\}$; $Q' = \emptyset$; $S' = (S \setminus \{a\}) \cup Q$;
ElseIf Q is a clique **then** *(2)*
  Let q be a vertex in Q;
  $F = \{av \mid v \in (C \setminus N(a)) \cup Q \setminus \{q\}\}$;
  $C' = C \cup \{a\} \cup (Q \setminus \{q\})$; $Q' = \emptyset$; $S' = (S \setminus \{a\}) \cup \{q\}$;
ElseIf $Q = \emptyset$ **then**
  $F = \{av \mid v \in C \setminus N(a)\}$;
  $W = \{x \mid x \in N(a) \cap C \wedge d(x) = \omega\}$;
  If $|W| = 0$ **then** *(3a)*
    $C' = C \cup \{a\}$; $Q' = \emptyset$; $S' = S \setminus \{a\}$;
  ElseIf $|W| > 1$ **then** *(3b)*
    $C' = (C \setminus W) \cup \{a\}$; $Q' = W$; $S' = S \setminus \{a\}$;
  ElseIf $W = \{c\}$ **and** $d(b) = \omega - 1$ **then** *(3c)*
    $C' = (C \setminus \{c\}) \cup \{a\}$; $Q' = \{b, c\}$; $S' = S \setminus \{a, b\}$;
  ElseIf $W = \{c\}$ **and** $d(b) < \omega - 1$ **then** *(3d)*
    $C' = (C \setminus \{c\}) \cup \{a\}$; $Q' = \{c\}$; $S' = S \setminus \{a\}$;
  EndIf
EndIf
$G' = (V,\ E \cup \{ab\} \cup F)$;

Lemma 25 *Given a split graph $G = (V, E)$ together with its 3-partition, and a pair of vertices $a, b \in V$ such that $ab \notin E$, Algorithm Minimal-edge-completion computes a minimal split completion $G' = (V, E \cup \{ab\} \cup F)$ of $G_{ab} = (V, E \cup \{ab\})$ and the 3-partition of G'.*

Proof. The cases of this algorithm can be easily proved if we consider the equivalent problem of adding a new vertex a to the split graph $G[V \setminus \{a\}]$, and use the Algorithm Minimal-vertex-completion from Section 3.2.3, where we compute a minimal split completion of $G[V \setminus \{a\}]_a = (V,\ E \cup \{ay \mid y \in N_G(a) \cup \{b\}) = G_{ab}$. It is then straightforward to verify that all fill edges have both endpoints inside C', meaning that the completion is minimal by Theorem 8.

Let $V \setminus \{a\} = S_a + C_a + Q_a$ be the 3-partition of $G[V \setminus \{a\}]$. Let us analyze each possible case. Remember that both a and b belong to S, since G_{ab} is not split.

Assume that Q is an independent set with at least two vertices (or we consider it a clique). In this case the 3-partition of $G[V \setminus \{a\}]$ is equal to that of G, thus $C_a = C, Q_a = Q, S_a = S \setminus \{a\}$. So $N_{G_{ab}}(a) \cap S_a \neq \emptyset$ and $N_{G_{ab}}(a) \cap Q_a = \emptyset$, therefore we add edges to make a adjacent to all vertices of C_a, according to Algorithm Minimal-vertex-completion. To do this, we simply let $F = \{av \mid v \in C_a \wedge av \notin E\} = \{av \mid v \in C \setminus N(a)\}$. Then $C' = C_a \cup \{a\}$, $Q' = \emptyset$, and $S' = S_a \cup Q_a$. Hence $C' = C \cup \{a\}$, $Q' = \emptyset$, and $S' = (S \setminus \{a\}) \cup Q$.

Assume that Q is a clique. In this case, some vertices in C could move to Q_a, namely a subset of $N_{G_{ab}}(a) \cap C$. Thus $C_a = C \setminus X, Q_a = Q \cup X$, and $S_a = S \setminus \{a\}$, where $X = \{x \mid x \in N_G(a) \cap C \wedge d_G(x) = \omega\}$. So $N_{G_{ab}}(a) \cap S_a \neq \emptyset$, and it might be that $N_{G_{ab}}(a) \cap Q_a \neq \emptyset$. We then make a adjacent to all vertices of C_a and to $|Q_a| - 1$ vertices in Q_a, according to Algorithm Minimal-vertex-completion. Let q be the

single vertex of Q which we do not make adjacent to a. We let $F = \{av \mid v \in (C_a \cup Q_a \setminus \{q\}) \wedge av \notin E\} = \{av \mid v \in (C \setminus N_G(a)) \cup Q \setminus \{q\}\}$ and $G' = (V,\ E \cup \{ab\} \cup F)$. Then $C' = C_a \cup \{a\} \cup (Q_a \setminus \{q\})$, $Q' = \emptyset$, and $S' = S_a \cup \{q\}$. Hence $C' = C \cup \{a\} \cup (Q \setminus \{q\})$, $Q' = \emptyset$, and $S' = (S \setminus \{a\}) \cup \{q\}$. We can also notice that if $|Q| = 1$, then case 1 and 2 are actually equivalent, even though we proved case 1 only for $|Q| > 1$.

Assume that $Q = \emptyset$. Thus G has a unique split partition. We will distinguish between the following cases.

1. If all neighbors of a in G that belong to C have degrees greater than ω, then $G[V \setminus \{a\}]$ will also have a unique split partition with $C_a = C, Q_a = Q = \emptyset, S_a = S \setminus \{a\}$. So $N_{G_{ab}}(a) \cap S_a \neq \emptyset$ and $N_{G_{ab}}(a) \cap Q_a = \emptyset$. Therefore it is enough to make a adjacent to all vertices in C_a. We simply let $F = \{av \mid v \in C_a \wedge av \notin E\} = \{av \mid v \in C \setminus N_G(a)\}$. In this case the maximum clique size increases by one, but since all vertices in C_a either had already a degree greater than ω or increase their degrees by one, they remain in C' and a moves to C' as well since it is adjacent to all vertices of C' and to b. Hence $C' = C_a \cup \{a\}$, $Q' = \emptyset$, and $S' = S_a$. And $C' = C \cup \{a\}$, $Q' = \emptyset$, and $S' = S \setminus \{a\}$.

2. If a has more than one neighbor in C with degree equal to ω, they will all move to Q_a (that becomes a clique), thus $C_a = C \setminus W, Q_a = W, S_a = S \setminus \{a\}$, where $W = \{x \mid x \in N_G(a) \cap C \wedge d_G(x) = \omega\}$. So $N_{G_{ab}}(a) \cap S_a \neq \emptyset$ and $N_{G_{ab}}(a) \cap Q_a = Q_a$. In this case a has already enough neighbors in Q_a, therefore it is still sufficient to make a adjacent to all C_a. Thus we have $F = \{av \mid v \in C_a \wedge av \notin E\} = \{av \mid v \in C \setminus N_G(a)\}$. Accordingly, $C' = C_a \cup \{a\}$, $Q' = W$, and $S' = S_a$. Hence $C' = (C \setminus W) \cup \{a\}$, $Q' = W$, and $S' = S \setminus \{a\}$.

3. In the special case in which there is exactly one $c \in C$ such that $N_G(c) \cap S = \{a\}$ (so $d_G(c) = \omega$) and $d_G(b) = \omega - 1$, then Q_a will be an independent set consisting of b, c and other vertices of S with degree $\omega - 1$. Thus $C_a = C \setminus \{c\}, Q_a = X \cup \{c\}, S_a = (S \setminus X) \setminus \{a\}$ where $X = \{x \mid x \in (S \setminus \{a\}) \wedge d_G(x) = \omega - 1\}$. So this is the only case in which $N_{G_{ab}}(a) \cap S_a = \emptyset$. Then, we need to make a adjacent to all vertices of C_a, since $N_{G_{ab}} \cap Q_a = 2$. We let $F = \{av \mid v \in C_a \wedge av \notin E\} = \{av \mid v \in C \setminus N_G(a)\}$. Then $C' = C_a \cup \{a\}$, $Q' = \{b, c\}$, and $S' = (S_a \setminus \{b\}) \cup (Q_a \setminus \{c\})$. Hence $C' = (C \setminus \{c\}) \cup \{a\}$, $Q' = \{b, c\}$ and $S' = S \setminus \{a, b\}$.

4. If there is exactly one c in C such that $N_G(c) \cap S = \{a\}$ (so $d_G(c) = \omega$), and there are some vertices in S_a with degree $\omega - 1$, but $d_G(b) < \omega - 1$, then Q_a will be an independent set consisting of c and such vertices of S except b. Thus $C_a = C \setminus \{c\}, Q_a = X \cup \{c\}, S_a = (S \setminus X) \setminus \{a\}$, where $X = \{x \mid x \in (S\{a\}) \wedge d_G(x) = \omega - 1\}$. So we have again $N_{G_{ab}}(a) \cap S_a \neq \emptyset$. Still it is sufficient to make a adjacent to all vertices of C, according to Algorithm Minimal-vertex-completion. We let $F = \{av \mid v \in C_a \wedge av \notin E\} = \{av \mid v \in C \setminus N_G(a)\}$. Then $C' = C_a \cup \{a\}$, $Q' = \{c\}$, and $S' = S_a \cup (Q_a \setminus \{c\})$. Hence $C' = C \setminus \{c\} \cup \{a\}$, $Q' = \{c\}$ and $S' = S \setminus \{a\}$.

■

Theorem 26 *Given a split graph $G = (V, E)$ together with its 3-partition, and a pair of vertices $a, b \in V$ such that $ab \notin E$, a minimal split completion G' of $G_{ab} = (V, E \cup \{ab\})$ and the 3-partition of G' can be computed in time $O(\max\{d_{G'}(a), d_{G'}(b)\})$.*

Proof. If G_{ab} is not a split graph, then we use the algorithm Minimal-edge-completion to compute a set to fill edges F and the 3-partition of the split graph $G' = (V, E \cup \{ab\} \cup F)$. So we need to prove that the algorithm Minimal-edge-completion can compute any output in time $O(\max\{d_{G'}(a), d_{G'}(b)\})$. Given the set W, each **If** statement can be check in $O(1)$, so we need to prove that to find W and perform each update can be done in the claimed running time. Notice that W can be found trivially in time $O(|N_G(a)|)$, just scanning the neighborhood of a.

The split completion consists in making one of the endpoints of ab completely adjacent to C and possibly $|Q| - 1$ vertices of Q, when Q is a clique, adding to it a set of fill edges F. Once we decide which endpoint to complete, let us say a, since both a and b belongs to S, in G' we have that $d_{G'}(a) = d_G(a) +$

$|F| \geq |C| > d_G(b) = d_{G'}(b)$, therefore it is always true that $\max\{d_{G'}(a), d_{G'}(b)\} = d_{G'}(a) = d_G(a) + |F|$. So proving that the algorithm runs in time $O(\max\{d_{G'}(a), d_{G'}(b)\})$, is equivalent to proving that it runs in time $O(d_G(a) + |F|)$ or $O(d_G(b) + |F|)$, according to which endpoint we complete.

Let us assume for simplicity that we chose to complete a.

Computing and adding F takes $O(|N_G(a)| + |F|) = O(d_G(a) + |F|)$ time. In fact we always need to make all vertices of C adjacent to a, meaning that we have to scan the set C to find the vertices that are not neighbors of a, and this takes $O(|C|)$, but $|C| \leq |N_G(a)| + |F|$, so this operation takes $O(|C|) = O(|N_G(a)|+|F|) = O(d_G(a)+|F|)$ time. Sometimes we also need to scan Q, taking $O(|Q|)$ time, but then we will make a adjacent to all vertices of Q except one vertex, meaning that $|C|+|Q|-1 = |N_G(a)|+|F|$, so that $O(|C| + |Q|) = O(|N_G(a)| + |F|) = O(d_G(a) + |F|)$.

To update the 3-partition after we added F to the graph, will take time $O(d_G(a) + |F|)$ as well.

First of all, notice that all cases where we move only a or b, or both, can be performed in constant time.

In case 1, we can perform the update in $O(1)$ time by lemma 11, because we move $Q \setminus A$ to S where $A = \emptyset$, and there cannot be vertices in S with degree $\omega(G) - 1$ because the partition of G is not unique.

In case 2, we need to move $Q \setminus \{q\}$ to C, but $Q \setminus \{q\} \subset N_{G'}(a)$, so we can perform this update in time $O(|N_{G'}(a)|) = O(d_G(a) + |F|)$. After we moved $Q \setminus \{q\}$ to C, $Q = \{q\}$, so it takes constant time to move q to S. In case 3b, as in the previous one, the set of vertices to be moved is a subset of $N_{G'}(a)$, so we can perform this operation in $O(|N_{G'}(a)|) = O(d_G(a) + |F|)$ time.

We can conclude that the algorithm Minimal-edge-completion runs in time $O(d_G(a) + |F|)$ as we claimed. ∎

3.3.4 Minimum split completion when the answer to edge addition query is no

In the proof of lemma 25, we showed how to formulate an edge addition as a special vertex addition. Using those cases it is straightforward to find the minimum fill for edge addition, since we can apply directly the formulas from Theorem 19.

Theorem 27 *Given a split graph $G = (V, E)$ together with its 3-partition $V = S + C + Q$, and a pair of vertices $a, b \in V$ such that $ab \notin E$, a minimum split completion G' of $G_{ab} = (V, E \cup \{ab\})$ can be computed by Algorithm Minimal-vertex-completion, choosing to complete the endpoint of ab with highest degree.*

Proof.

We know that G_{ab} is not split if and only if $ab \in S$. Algorithm Minimal-vertex-completion cover all cases in which $ab \in S$, so we can formulate every case as a vertex addition, and apply the formulas from Theorem 19. This will show that a minimum split completion in this case can be obtained simply by adding to the graph the smallest between a set of fill edges F_1, consisting of edges incident only to a, and F_2, consisting of edges incident only to b. Besides these sets of fill edges are the same as Algorithm Minimal-vertex-completion would compute. Since it will be easy to see that checking which the smallest is, corresponds to checking the minimum between $|C \setminus N_G(a)|$ and $|C \setminus N_G(b)|$, it follows that we always choose the endpoint of maximum degree as claimed.

We use the same notation as in the proof of Lemma 25, thus $G[V \setminus \{a\}]_a = (V,\ E \cup \{ay \mid y \in N_G(a) \cup \{b\}) = G_{ab}$, and its 3-partition is $V_a = S_a + C_a + Q_a$. Besides, since we will apply the formulas of Theorem 19 to $G[V \setminus \{a\}]_a$, we replace x with a, N_x with $N_{G_{ab}}(a)$ and the 3-partition $V = S + C + Q$, with $V_a = S_a + C_a + Q_a$.

- In case 1 of Algorithm Minimal-vertex-completion, $C_a = C, Q_a = Q, S_a = S \setminus \{a\}$ and Q_a is an independent set. Hence we must apply the formulas of case 2 of Theorem 19 with $N_{G_{ab}}(a) \cap S_a = \{b\}$ and $N_{G_{ab}}(a) \cap (S_a \cup Q_a) = \{b\}$, and we get: $F_1 = \{av \mid v \in C_a \setminus N_{G_{ab}}(a)\} = \{av \mid v \in C \setminus N_G(a)\}$, $F_2 = \{bv \mid v \in C_a \setminus N_{G_{ab}}(b)\} = \{bv \mid v \in C \setminus N_G(b)\}$, and $|F_{min}| = \min\{|C \setminus N_G(a)|, |C \setminus N_G(b)|\}$.

- In case 2 of Algorithm Minimal-vertex-completion, $C_a = C \setminus X, Q_a = Q \cup X$, and $S_a = S \setminus \{a\}$, and Q_a is a clique where $X = \{x \mid x \in N_G(a) \cap C \wedge d_G(x) = \omega\}$. Hence we must apply formulas of case 1 of Theorem 19 with $|N_{G_{ab}}(a) \cap S_a| = \{b\}$ and $0 \leq |N_{G_{ab}}(a) \cap Q_a| < |Q_a|$, and we get: $F_1 = \{av \mid v \in (C_a \cup Q_a \setminus \{q\}) \setminus N_{G_{ab}}(a)\} = \{av \mid v \in (C \cup Q \setminus \{q\}) \setminus N_G(a)\}$, where q is a vertex of Q, and $F_2 = \{bv \mid v \in (C_a \cup Q_a \setminus \{q\}) \setminus N_{G_{ab}}(b)\} = \{bv \mid v \in (C \cup Q \setminus \{q\}) \setminus N_G(b)\}$, and $|F_{min}| = \min\{|(C \cup Q) \setminus N_G(a)| - 1, |(C \cup Q) \setminus N_G(b)| - 1\}$, but since $Q \setminus N_G(a) = Q \setminus N_G(b) = Q$, we can find the minimum set of filll edges comparing only $|C \setminus N_G(a)|$ and $|C \setminus N_G(b)|$.

- In case 3a of Algorithm Minimal-vertex-completion, $C_a = C, Q_a = Q = \emptyset, S_a = S \setminus \{a\}$. Hence we must apply formulas of case 3 of Theorem 19 with $|N_{G_{ab}}(a) \cap S_a| = \{b\}$, so we get: $F_1 = \{av \mid v \in C_a \setminus N_{G_{ab}}(a)\} = \{av \mid v \in C \setminus N_G(a)\}$, $F_2 = \{bv \mid v \in C_a \setminus N_{G_{ab}}(b)\} = \{bv \mid v \in C \setminus N_G(b)\}$, and $|F_{min}| = \min\{|C \setminus N_G(a)|, |C \setminus N_G(b)|\}$.

- In case 3b of Algorithm Minimal-vertex-completion, $C_a = C \setminus W, Q_a = W, S_a = S \setminus \{a\}$, where $W = \{x \mid x \in N_G(a) \cap C \wedge d_G(x) = \omega\}$. Hence we must apply formulas of case 1 of Theorem 19 with $|N_{G_{ab}}(a) \cap S_a| = \{b\}$ and $N_{G_{ab}}(a) \cap Q_a = Q_a$, so we get: $F_1 = \{av \mid v \in C_a \setminus N_{G_{ab}}(a)\} = \{av \mid v \in C \setminus N_G(a)\}$, since $Q_F = \emptyset$, and $F_2 = \{bv \mid v \in (C_a \cup Q_a) \setminus N_{G_{ab}}(b)\} = \{bv \mid v \in C \setminus N_G(b)\}$, since $N_{G_{ab}}(b) \cap Q_a = \emptyset$ and $C = C_a \cup Q_a$. Besides $|F_{min}| = \min\{|C \setminus N_G(a)|, |C \setminus N_G(b)|\}$.

- In case 3c of Algorithm Minimal-vertex-completion, $C_a = C \setminus \{c\}, Q_a = X \cup \{c\}, S_a = (S \setminus X) \setminus \{a\}$ where $X = \{x \mid x \in (S \setminus \{a\}) \wedge d_G(x) = \omega - 1\}$ where c is the only vertex in $C \cap N_G(a)$ with degree ω. Hence, since $N_{G_{ab}}(a) \cap S_a = \emptyset$, we must apply formulas of case 4 of Theorem 19 with $N_{G_{ab}}(a) \cap Q_a = \{c, b\}$, so we get: $F_1 = \{av \mid v \in C_a \setminus N_{G_{ab}}(a)\} = \{av \mid v \in C \setminus N_G(a)\}$, since $C = C_a \cup \{c\}$, but c is already adjacent to a, and $F_2 = \{bv \mid v \in N_{G_{ab}}(a) \cap Q_a\} = \{bv \mid v \in C \setminus N_G(b)\}$, because $N_{G_{ab}}(a) \cap Q_a = \{c, b\}$ and $c = C \setminus N_G(b)$. Again $|F_{min}| = \min\{|C \setminus N_G(a)|, |C \setminus N_G(b)|\}$.

- In case 3d of Algorithm Minimal-vertex-completion, $C_a = C \setminus \{c\}, Q_a = X \cup \{c\}, S_a = (S \setminus X) \setminus \{a\}$, where $X = \{x \mid x \in (S \setminus \{a\}) \wedge d_G(x) = \omega - 1\}$ and c is the only vertex in $C \cap N_G(a)$ with degree ω. Hence we must apply formulas of case 2 of Theorem 19 with $|N_{G_{ab}}(a) \cap S_a| = \{b\}$ and $N_{G_{ab}}(a) \cap Q_a = \{c\}$, so we get: $F_1 = \{av \mid v \in C_a \setminus N_{G_{ab}}(a)\} = \{av \mid v \in C \setminus N_G(a)\}$, since $C = C_a \cup \{c\}$ and $c \in N_{G_{ab}}(a)$, and $F_2 = \{bv \mid v \in (C_a \cup \{c\}) \setminus N_{G_{ab}}(b)\} = \{bv \mid v \in C \setminus N_G(b)\}$, so that $|F_{min}| = \min\{|C \setminus N_G(a)|, |C \setminus N_G(b)|\}$.

■

Theorem 28 *Given a split graph $G = (V, E)$ together with its 3-partition $V = S + C + Q$, and a pair of vertices $a, b \in V$ such that $ab \notin E$, a minimum split completion G' of $G_{ab} = (V, E \cup \{ab\})$ can be computed in $O(\max\{d_{G'}(a), d_{G'}(b)\})$ time.*

Proof. The running time follows from Theorem 27 and 26, since we can find the endpoint of highest degree in constant time. ■

3.4 Deleting an edge from G

In this section, we describe the query and update operations for deleting an edge ab from a given split graph $G = (V, E)$. A query for deleting this edge is simply deciding whether $G^-_{ab} = (V, E \setminus \{ab\})$ is a split graph. An update if the answer to this query is yes requires simply to update the 3-partition $V = S + C + Q$ of G so that it becomes a 3-partition of G^-_{ab}. We can perform both a query and an update in $O(1)$ time, matching the running time given by Ibarra [12] for these operations.

In case the query replies that G^-_{ab} is not split, then the update query has the choice of deleting a minimal or a minimum set of additional edges from G^-_{ab} to obtain a split graph. Computing minimal or minimum split deletions of G^-_{ab} have not been studied before. We could of course remove the edge ab, and then do a minimal or minimum split completion of the complement of the resulting graph, but doing this in a straight forward way would result in $O(n^2)$ running time. We are able to implement these operations in time $O(\min\{d_G(a), d_G(b)\})$.

3.4.1 Query for deleting an edge

Lemma 29 *Given a split graph $G = (V, E)$, its 3-partition is $V = S + C + Q$, and an edge $ab \in E$, $G^-_{ab} = (V, E \setminus \{ab\})$ fails to be a split graph if and only if $a, b \in C$.*

Proof. Let $\bar{G}$ be the complement of G and let $V = \bar{S} + \bar{C} + \bar{Q}$ be the 3-partition of $\bar{G}$. By Corollary 10, $\bar{S} = C, \bar{C} = S, \bar{Q} = Q$. Since deleting an edge from G is equivalent to adding an edge to $\bar{G}$, by Theorem 21 we can add an edge to $\bar{G}$ everywhere except between to vertices that belong to $\bar{S}$, meaning that we can delete any edge from G except those that have both endpoints in C. ∎

Thus to answer the query for edge addition, we only need to check whether both a and b belong to C, and the constant time bound follows.

Theorem 30 *Given a split graph $G = (V, E)$, its 3-partition, and an edge $ab \in E$, it can be decided in $O(1)$ time whether $G^-_{ab} = (V, E \setminus \{ab\})$ is a split graph.*

3.4.2 Update when the answer to edge deletion query is yes

Algorithm: Update-edge-deletion

Input: A split graph $G = (V, E)$, with 3-partition $V = S + C + Q$ and an edge $ab \in E$.
Output: A split graph $G^-_{ab} = (V, E \setminus \{ab\})$ and its with 3-partition $V = S' + C' + Q'$.

$d(b) = |N_G(b)|$;
$\omega = \omega(G)$;
If $a \in S$ and $b \in C$ **then**
  If $d(b) = \omega$ **and** $Q = \emptyset$ **then** *(1a)*
    $W = \{v \in S \mid d(v) = \omega - 1\}$;
    $S' = S \setminus W$; $C' = C \setminus \{b\}$; $Q' = Q \cup W \cup \{b\}$;
  ElseIf $d(b) = \omega$ **and** $Q \neq \emptyset$ **then** *(1b)*
    $S' = S$; $C' = C \setminus \{b\}$; $Q' = Q \cup \{b\}$;
  ElseIf $d(b) > \omega$ **then** *(1c)*
    $S' = S$; $C' = C$; $Q' = Q$;
  EndIf
ElseIf $a \in Q$ and $b \in Q$ **then**
  If $|Q| = 2$ **then** *(2a)*
    $W = \{v \mid v \in S \wedge d(v) = \omega - 2\}$;
    $S' = S \setminus W$; $C' = C$; $Q' = Q \cup W$;
  ElseIf $|Q| > 2$ **then** *(2b)*
    $S' = S$; $C' = C \cup Q \setminus \{a, b\}$; $Q' = \{a, b\}$;
  EndIf
ElseIf $a \in Q$ and $b \in C$ **then**
  If Q is a clique **then** *(3a)*
    $S' = S \cup \{a\}$; $C' = C \cup Q \setminus \{a\}$; $Q' = \emptyset$;
  ElseIf Q is an independent set of size greater than 1 **then**
    If $d(b) = \omega$ **then** *(3b1)*
      $S' = S \cup \{a\}$; $C' = C \setminus \{b\}$; $Q' = (Q \cup \{b\}) \setminus \{a\}$;
    ElseIf $d(b) > \omega$ **then** *(3b2)*
      $S' = S \cup \{a\}$; $C' = C$; $Q' = Q \setminus \{a\}$;
    EndIf
  EndIf
EndIf
$G^-_{ab} = (V, E \setminus \{ab\})$;

Lemma 31 *Given a split graph $G = (V, E)$, an edge $ab \in E$, and the knowledge that $G^-_{ab} = (V, E \setminus \{ab\})$ is a split graph, Algorithm Update-edge-deletion computes the 3-partition of G^-_{ab}.*

Proof. It is easy to check that the algorithm covers all possible cases. We will show that what the algorithm does in each case is correct.

When we remove an edge ab between S and C, we need to distinguish between three cases according to the degree of the endpoint in C, namely b. Removing this edge cannot change the size of the maximum clique, so a will remain in S in any case, but if $d_G(b) = \omega(G)$, it will be $d_{G_{ab^-}}(b) = \omega(G) - 1$, so b must be moved to Q. Hence: If $d_G(b) > \omega(G)$, the 3-partition does not change (case 1c); If $Q \neq \emptyset$, all vertices of degree $\omega(G) - 1$ are already in Q, so we need to move only b from C to Q (case 1b); and if $Q = \emptyset$, since we move b from C and the partition is not unique anymore, if there are vertices of degree $\omega(G) - 1$ in S, they must be moved in Q as well (case 1a).

When we remove an edge ab in Q (case 2a and 2b), the size of the maximum clique always decreases by one, so $\omega' = \omega(G^-_{ab}) = \omega - 1$, so a and b remain in Q because after deleting the edge ab their degrees become $\omega(G) - 2 = \omega' - 1$. All other vertices of $Q \setminus \{a, b\}$ must be moved to C since their degree is still $\omega(G) - 1 = \omega'$. This means that in G^-_{ab}, $|C'| = \omega' - 1$, so the partition is not unique, and all vertices with degree $\omega(G) - 2$ in S, that have therefore degree $\omega' - 1$ in G^-_{ab}, must be moved to Q. However notice that there can be vertices with degree $\omega(G) - 2$ in S only if $|Q| = 2$ (case 2a).

Removing an edge ab between Q and C, the size of the maximum clique decreases by one only if Q is a clique (case 3a), so $\omega' = \omega(G^-_{ab}) = \omega - 1$ and all vertices in $Q \setminus \{a\}$ must be moved to C, since they have degree $\omega(G) - 1 = \omega'$. This means that in G^-_{ab}, $|C'| = \omega'$ and the partition is unique, so a goes to S although its degree is $\omega' - 1$.

Removing the edge ab between Q and C, does not change the size of the maximum clique if Q is an independent set with more than one vertex (or it can be considered as a clique). This means that nothing is affected except the endpoints of ab. The endpoint in Q, namely a, has degree $\omega' - 2$ in G^-_{ab}, so it must always be moved to S. The endpoint in C, namely b, must be moved to Q only if $d_G(b) = \omega$, because after removing ab, it would have degree $\omega - 1 = \omega' - 1$ (case 3b1). ∎

Theorem 32 *Given a split graph $G = (V, E)$ together with its 3-partition, an edge $ab \in E$, and the knowledge that $G^-_{ab} = (V, E \setminus \{ab\})$ is split, the 3-partition of G^-_{ab} can be computed in time $O(1)$.*

Proof. To prove the theorem we need to prove that the update of the 3-partition in every case of Algorithm Update-edge-deletion takes time $O(1)$.

To evaluate each **If** statement takes $O(1)$ time, in fact we only need to check wich set the endpoints of ab are in, their degree, and which type of set Q is.

In case 1a, the set W, is exactly the degree list $L_{\omega(G)-1}$ of S, so it can be found in $O(1)$ time. Once we are given W, by Lemma 11 it takes constant time to move it to Q and update the structure, in fact we can consider $A = \emptyset$, $W = X_d$, and we know that there are no vertices with the same degree as W in Q, because it is empty in G by assumption.

Using the same argument, we can prove that it takes constant time to update the 3-partition in case 2a as well, with the only difference that Q is not empty, but there is a constant number of vertices in it.

In case 2b and 3a, we need to move $Q \setminus A$ to C, where $A = \{a, b\}$ and $A = \{a\}$ respectively. Since by definition there cannot be vertices of degree $\omega(G) - 1$ in C, we can apply Lemma 11, so that the update in these cases takes $O(|A|) = O(1)$ time.

In all remaining cases we only need to move a or b, or both, and such updates require constant time as well, so the theorem follows. ∎

3.4.3 Minimal split deletion when the answer to edge deletion query is no

If the answer to edge deletion query is no, then it is possible to delete a minimal set of edges from one of the endpoints of the edge we want to delete, to make the graph split again.

Algorithm: Minimal-edge-deletion

Input: A split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, and an edge $ab \in E$
Output: A minimal split deletion (maximal split subgraph) G' of $G^-_{ab} = (V, E \setminus \{ab\})$ and the 3-partition $V = S' + C' + Q'$ of G'.

$d(b) = |N_G(b)|$;
$\omega = \omega(G)$;
If Q is a clique **then** *(1)*
  $F^- = \{av \mid v \in S \wedge av \in E\}$;
  $S' = S \cup \{a\}$; $Q' = \emptyset$; and $C' = C \setminus \{a\} \cup Q$;
ElseIf Q is an independent set **then** *(2)*
  Let q be a vertex in Q;
  $F^- = \{av \mid v \in (S \cup Q \setminus \{\ q\}) \wedge av \in E\}$;
  $S' = S \cup \{a\} \cup (Q \setminus \{q\})$; $Q' = \emptyset$; and $C' = (C \setminus \{a\}) \cup \{q\}$;
ElseIf Q is empty **then**
  $F^- = \{av \mid v \in S \wedge av \in E\}$;
  $W = \{x \mid x \in S \setminus N_G(a) \wedge d_G(x) = \omega - 1\}$;
  If $|W| = 0$ **then** *(3a)*
    $S' = S \cup \{a\}$; $Q' = \emptyset$; $C' = C \setminus \{a\}$;
  ElseIf $|W| > 1$ **then** *(3b)*
    $S' = S \setminus W \cup \{a\}$; $Q' = W$;$C' = C \setminus \{a\}$;
  ElseIf $W = \{s\}$ **and** $d(b) = \omega$ **then** *(3c)*
    $S' = (S \setminus \{s\}) \cup \{a\}$; $Q' = \{b, s\}$; $C' = C \setminus \{a, b\}$;
  ElseIf $W = \{s\}$ **and** $d(b) > \omega$ **then** *(3d)*
    $S' = S \setminus \{s\} \cup \{a\}$; $Q' = \{s\}$; $C' = C \setminus \{a\}$;
  EndIf
EndIf
$G' = (V, E \setminus \{ab\} \setminus F^-)$;

Lemma 33 *Given a split graph $G = (V, E)$ together with its 3-partition and an edge $ab \in E$, Algorithm Minimal-edge-deletion computes a maximal split deletion G' of $G^-_{ab} = (V, E \setminus \{ab\})$ and the 3-partition of G'.*

Proof. To prove the correctness of the algorithm, it is enough to check that each case is the correct translation of the equivalent case in algorithm Minimal-edge-completion, when we want to add the edge ab to the complement graph $\bar{G}$ with partition $V = \bar{S} + \bar{C} + \bar{Q}$. To do this we use Lemma 9 and Corollary 10. We refer to vertex a, but by symmetry, every operation can be performed on b as well. Remember that $ab \in C$ by Lemma 29, so $ab \in \bar{S}$ in $\bar{G}$ as expected.

Once we find to which case of Algorithm Minimal-edge-completion a case of Algorithm Minimal-edge-deletion corresponds, there are two things that must be checked to guarantee the correctness of such cases. The first is that adding F to $\bar{G}$, we delete exactly F^- from G. The second is that, applying corollary 10 to the 3-partition of $\bar{G}'$ given in Algorithm Minimal-edge-completion, we get exactly the 3-partition of G' given for the corresponding case in Algorithm Minimal-edge-deletion. However, since this can be done straightforwardly comparing the two algorithms, we only show the correspondence between cases.

When Q is a clique (case 1), then $\bar{Q}$ is an independent set, since $Q = \bar{Q}$, then we apply case 1 of Minimal-edge-completion to $\bar{G}$.

When Q is an independent set (case 2), then $\bar{Q}$ is a clique, since $Q = \bar{Q}$, then we apply case 2 of Minimal-edge-completion to $\bar{G}$.

When $Q = \emptyset$, $\bar{Q} = \emptyset$ as well, so we need to check the degree of b in $\bar{G}$ and which vertices belong to the set W. Since $Q = \emptyset$, it follows that $\bar{\omega} = \omega(\bar{G}) = |S|$, and for a vertex $s \in S$, we have $d_{\bar{G}}(s) = |S \setminus \{s\}| + |C \setminus N_G(s)| = \bar{\omega} - 1 + |C \setminus N_G(s)|$. This means that the vertices of S not incident to a and with degree $\omega - 1$ in G, are exactly the vertices of $\bar{G}$ that are in $\bar{C}$, are adjacent only to a and have

therefore degree $\bar{\omega}$. So the set W in Algorithm Minimal-edge-completion is exactly the set W in the algorithm Minimal-edge-deletion. Besides, when $d_G(b) = \omega$, then $d_{\bar{G}}(b) = \omega - 1$, and when $d_G(b) > \omega$ then $d_{\bar{G}}(b) < \omega - 1$.

We conclude that cases 3a, 3b, 3c and 3d of Algorithm Minimal-edge-completion, correspond exactly to cases 3a, 3b, 3c and 3d of Algorithm Minimal-edge-deletion. ∎

Theorem 34 *Given a split graph $G = (V, E)$ together with its 3-partition and an edge $ab \in E$, a minimal split deletion G' of $G^-_{ab} = (V, E \setminus \{a, b\})$ and the 3-partition of G' can be computed in time $O(\min\{d_G(a), d_G(b)\})$.*

Proof.

First of all we assume that we choose to delete edges only incident to the endpoint a, keeping in mind that everything still holds if we choose b. Now we need to to prove that every case of Algorithm Minimal-edge-deletion can be executed in $O(d_G(a))$ time. Remember that both $a, b \in C$ so they are adjacent to all vertices of $C \cup Q$. To find F^- takes $O(d_G(a))$ time, because we always delete edges adjacent to a, and to find the right ones, we only need to scan the neighborhood of a in G.

In case 1a and 2a, we can move Q respectively to C or S, in $O(d_G(a))$ time, because $|Q| = O(|N_G(a)|)$. Notice that in cases 3a, 3b, 3c and 3d, F^- is the same, so it does not depend on the **If** condition. Hence, in $O(d_G(a))$ time, we can find F^-, remove these edges from G, and update the degree lists of the sets of the 3-partition accordingly, without moving any vertex from a set to another. Now the 3 sets Q, S and C are not a correct 3-partition of G', but we need to move only W, a and possibly b, to other sets. However, F^- is exactly the set of edges from a to S, and since we removed them and updated all degree lists, now there cannot be vertices with degree $\omega(G) - 1$ in S that were neighbors of a. Hence W is exactly the degree list $L_{\omega(G)-1}$ of the current S. This means that we can apply Lemma 11 to all cases 3a, 3b, 3c and 3d, updating the 3-partition in constant time, after we removed F^-.

Since we proved that each case can be executed in $O(d_G(a))$ time and we can choose in constant time the endpoint of ab with minimum degree, the theorem follows. ∎

3.4.4 Minimum split deletion when the answer to edge deletion query is no

Theorem 35 *Given a split graph $G = (V, E)$ together with its 3-partition, and an edge $ab \in E$, a minimum split deletion (split subgraph with the maximum number of edges) G' of $G^-_{ab} = (V, E \setminus \{ab\})$ can be computed in time $O(\min\{d_G(a), d_G(b)\})$.*

Proof. By symmetry with Theorem 27, it can be proven that the minimum deletion can be obtained choosing the endpoint with lowest degree of the edge that has been deleted. Hence the theorem follows from Theorem 34. ∎

3.5 Deleting a vertex from G

Since split graphs are perfect, every induced subgraph of a split graph is still split, hence it is always possible to remove a vertex from the graph, keeping it split. For this reason, in this section we give only a small algorithm that deletes a vertex and the edges incident to it one by one, keeping the graph split at each step, so that the edge deletion algorithm can be directly plugged in instead.

Given a split graph $G = (V, E)$ and a vertex $x \in V$, we first show that there is an order of the edges incident to x, such that if these edges are deleted in that order, the resulting graph remains split after each edge removal. We will call such an order a *good edge deletion order*.

Lemma 36 *Given a split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, and a vertex $x \in Q \cup S$, any order of the edges incident to x is a good edge deletion order.*

Proof. We know that there exists a split partition $V = I + K$ of G such that x is in I (there is always at least one vertex of Q that can be moved to I, so we can always choose x). Since all edges incident to x are then between the independent set and the clique, they can be removed in any order, and the graph will be split after each removal. ∎

Lemma 37 *Given a split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, and a vertex $x \in C$, the following is a good edge deletion order of the edges incident to x: First remove edges between x and its neighbors in S, then remove edges between x and its neighbors in Q, and finally remove edges between x and its neighbors in C.*

Proof. Let $v_0, v_1, ..., v_k$ be the neighbors of x in G, listed according to the edge deletion order described in the lemma. Let $G_0 = G$, and G_i be the result of deleting edge xv_{i-1} from G_{i-1}, and let $V = S_i + C_i + Q_i$ be the 3-partition of G_i, for $1 \leq i \leq k+1$. We have to show that when we delete edge xv_i from split graph G_i, the resulting graph G_{i+1} is split for each $i \in \{0, 1, ..., k\}$. By Lemma 29, this is equivalent to showing that x and v_i do not both belong to C_i.

Let us assume for the sake of contradiction that at some step i, both x and v_i belong to C_i, and let i be the first such step. In this case, we show that v_i belongs to $S \cup Q$. Since all vertices of $S \cup Q$ are ordered before all vertices of C among the endpoints of edges incident to x, if x has any neighbors in G_i belonging to $S \cup Q$, then $v_i \in S \cup Q$. Assume on the contrary that x has no neighbors belonging to $S \cup Q$ in G_i. Let $j < i$ be the first step where v_j belongs to C. Observe that the degree of each vertex of C remains unchanged until step j, and the size of the maximum clique can only have decreased. Thus every vertex of C belongs to C_j. Since in G_j, x has no neighbors outside of C_j, x does not belong to C_j. By Lemma 36, we can remove each remaining edge incident to x, contradicting the assumption that x and v_i both belong to C_i at some later step i. Thus we can conclude that $v_i \in S \cup Q$. It follows that $d_G(v_i) \leq \omega(G) - 1$, but since now v_i belongs to C_i, it means also that $d_{G_i}(v_i) \geq \omega(G_i)$ and $\omega(G_i) < \omega(G)$. On the other hand, the size of the maximum clique cannot decrease more than one since we remove edges incident only to one vertex, so $\omega(G_i) \geq \omega(G) - 1$. We can then conclude that $\omega(G_i) = \omega(G) - 1$ and therefore $d_G(v_i) = \omega(G) - 1$, so either $v_i \in S$ and $Q = \emptyset$, or $v_i \in Q$. In the first case, before step i, we have removed only edges incident to vertices of S, meaning that the size of the maximum clique cannot have decreased, so we have a contradiction. In the latter case, we also consider that since $x \in C_i$, it must have a neighbor y outside C_i. However $d_{G_i}(y) = d_G(y) < d_{G_i}(v_i) = d_G(v_i) = \omega(G) - 1$, meaning that $y \in S$ and we should have deleted the edge incident to y before the one incident to v_i, so we have a contradiction again. ∎

We are now ready to give the algorithm:

Algorithm: Update-vertex-deletion

Input: A split graph $G = (V, E)$, its 3-partition $V = S + C + Q$, and a vertex $x \in V$.
Output: A split graph $G' = G[V \setminus \{x\}]$, and its 3-partition $V \setminus \{x\} = S' + C' + Q'$

$G' = G$; $S' = S$, $C' = C$; $Q' = Q$;
$N = N_G(x)$;
Sort N according to a good edge deletion order;
While $(N \neq \emptyset)$
  Take the first vertex $v \in N$;
  $[G', S', C', Q'] =$ Update-edge-deletion(G', S', C', Q', xv);
  $N = N \setminus \{v\}$;
EndWhile
Delete the isolated vertex x from G';

Lemma 38 *Given a split graph $G = (V, E)$ together with its 3-partition and a vertex $x \in V$, Algorithm Update-vertex-deletion computes the 3-partition of $G[V \setminus \{x\}]$.*

Proof. It follows from the fact that there always exists a good edge deletion order, and that the Algorithm Update-edge-deletion is correct. ∎

Theorem 39 *Given a split graph $G = (V, E)$ together with its 3-partition and a vertex $x \in V$, the 3-partition of $G[V \setminus \{x\}]$ can be computed in time $O(d_G(x))$.*

Proof. It takes $O(d_G(x))$ to compute a good edge deletion order of $N_G(x)$. It is enough to create a list (or an array of size $d(x)$) and scan the adjacency list of x at most three times, adding to the list at each iteration the neighbors of x in one of the three sets S, Q, and C in this order. By theorem 32, it takes constant time to remove one edge from the graph using the algorithm Update-edge-deletion, and since we run such algorithm $d_G(x)$ times, the theorem follows. ∎

4 Vertex incremental algorithms for split graphs

The dynamic operations that we presented in the previous sections very easily lead to two interesting vertex incremental algorithms: A linear-time certifying algorithm for recognizing split graphs, and an algorithm for minimal split completions with running time linear in the size of the output graph. A linear-time certifying algorithm for recognizing split graphs was given by Heggernes and Kratsch [8], however that algorithm is not vertex incremental. An algorithm for minimal split completions was given by the authors [6], and that algorithm is not vertex incremental, either. The running time of the algorithm of [6] is linear in the size of the input graph, however only an implicit representation of the resulting split completion is output by that algorithm. Thus if the output graph is to be represented explicitly, the new incremental algorithm matches the time of the previous minimal split completion algorithm, as well. Vertex incremental algorithms have the advantage that the graph can be supplied one vertex at a time in an on-line fashion.

4.1 Vertex incremental certifying algorithm for recognition

An algorithm for recognizing a graph class is called *certifying* if it outputs a *certificate* together with its reply of yes or no regarding whether or not the given graph belongs to the class. The idea is then that this reply can be checked by a separate algorithm called an *authentication algorithm*. The authentication algorithm takes as input the input graph, the reply, and the certificate, and verifies that the reply is correct. In the case of membership, the certificate is usually a representation of the graph according to the definition of the graph class. In the case of non-membership, the certificate is usually a forbidden induced subgraph. A thorough discussion about certifying algorithms is given in [14].

Theorem 40 *There is a linear-time vertex incremental certifying algorithm for recognizing split graphs, such that the certificate for membership can be authenticated in $O(n + m)$ time and the certificate for non-membership can be authenticated in $O(n)$ time.*

Proof. Given a graph $G = (V, E)$, at each step we consider an induced subgraph $G[X]$ where $X \subseteq V$. In the beginning X contains an arbitrary vertex of V, and at each step X grows with one arbitrary vertex of V. Assuming that $G[X]$ is split, we check whether $G[X \cup \{x\}]$ is split by using the query operation for vertex addition at each step. Each such step takes $O(d(x))$ time by the results of Section 3.2. If the answer to the query is yes at each step, we can conclude at the step when $X = V$ that the input graph is split, and we can output a split partition or a 3-partition as a certificate. This takes clearly a total of linear time. If at some step, the answer to the query is no, we can stop, and find an induced subgraph of $G[X \cup \{x\}]$ that is a $2K_2$, C_4 or C_5 as described in the proof of Theorem 12. This will also be an induced subgraph of the input graph, and we output it as a certificate. Since there are a constant number of cases to check, this can be done in linear time after we conclude that the graph is not split, giving again a total of linear time for the whole algorithm.

In the case of membership, it can be checked in $O(n+m)$ time that the output split partition or 3-partition indeed corresponds to the claimed cliques and independent sets of the graph. In the case of non-membership the certificate is of constant size, and since every vertex of the certificate has at most $O(n)$ neighbors in the graph, checking that the certificate is one of the 3 forbidden subgraphs can be done in $O(n)$ time. ∎

4.2 Vertex incremental minimal split completion

If a graph class is hereditary and has the property that adding a universal vertex to a graph of the class always results in a graph which is also in the class, then minimal completions of arbitrary graphs into this class can be computed in a vertex incremental way [1, 7, 9]. This means that, for an input graph $G = (V, E)$ and a vertex set $X \subset V$, if we have a minimal completion H of $G[X]$, then a minimal completion of $G[X \cup \{x\}]$ can be computed by computing a minimal completion of $H_x = (X \cup \{x\},\ E(H) \cup \{xy \mid y \in N_{G[X\cup\{x\}]}(x)\})$ in such a way that each added fill edge is incident to x. Thus at each incremental step, the minimal completion of the previous step can be regarded as an input graph in the class to which we want to add a vertex with a given neighborhood, and we require fill edges to be incident to the incremental vertex x at each step.

Theorem 41 *There is a vertex incremental algorithm for computing minimal split completions of arbitrary graphs that has running time linear in the size of the returned split completion.*

Proof. This algorithm proceeds as the vertex incremental recognition algorithm. At each step if the reply to vertex addition query is yes, we use the corresponding update operation. If the reply to vertex addition query is no, we use the minimal completion operation. By the results of Section 3.2, each operation takes $O(d'(x))$ time where d' is the degree of a vertex in the output graph, and x is the vertex of the incremental step. Thus the total time is $(n + m + |F|)$ where F is the set of edges added to the input graph $G = (V, E)$ to obtain the output minimal split completion $(V, E \cup F)$. The correctness follows from the above discussion. ∎

Notice that using a minimum split completion as described in Section 3.2.4 in this algorithm, would not guarantee either minimality or linear time. In fact at some step the minimum fill could consists of edges not incident to the new vertex we are adding.

5 Split+1v, split+1e, and split−1e graphs

Split+1v, split+1e, and split−1e graphs can trivially be recognized in polynomial time. For membership in split+1v and split+1e of an input graph G, one can check for each vertex or edge of the graph, whether removing this vertex or edge results in a split graph. For membership in split−1e, one can check for each non-edge of the graph, whether adding this edge results in a split graph. In this section we will show that these classes can in fact be recognized in linear time. These results are not based on the dynamic operations of the previous sections. However, based on these results and the dynamic operations of the previous sections, we will be able to give linear time algorithms for computing minimum split completions of split+1v and split+1e graphs, and for computing minimum split deletions of split−1e graphs. Notice also that the proofs of Theorems 12, 21, 29, can be easily turned into linear time algorithms for producing a certificate that a given split graph plus a vertex or an edge, or minus an edge, is not split.

5.1 Certifying recognition of split+1e in linear time

Observation 42 *A split graph that is not edgeless is a split+1e graph.*

Proof. In a split graph $G = (V, E)$ with split partition $V = I + K$, any edge between K and I can be deleted to obtain a new split graph with one less edge. If I consists of isolated vertices, then any edge of K can be deleted to obtain a split graph with one less edge. ∎

Observation 43 *A split+1e graph does not contain C_5 as an induced subgraph.*

Proof. If adding an edge to a graph G results in a C_5 then this means that G contains a $2K_2$, implying that G is not split. Hence a graph obtained by adding an edge to a split graph cannot contain a C_5. ∎

Theorem 44 *There is a linear-time certifying algorithm for recognizing split+1e graphs, such that a certificate for membership can be authenticated in $O(n+m)$ time, and a certificate for non-membership can be authenticated in $O(n)$ time.*

Proof. First we run a certifying algorithm for recognizing split graphs. If the reply from this algorithm is yes, and the input graph is not edgeless, we return yes, and an edge of the graph according to the proof of Observation 42 as certificate. If the reply from the split recognition algorithm is no, then we examine the certificate for split non-membership. If the certificate is a C_5, we return no, and the C_5 as our certificate. If the certificate is a C_4 or a $2K_2$, then for each edge e of this induced subgraph, we check whether removing this edge results in a split graph. This can be done in $O(n+m)$ time with a certifying split recognition algorithm. If we conclude that the resulting graph is split, we return yes, and we return e and a split partition (or 3-partition) of the graph without edge e. If we conclude that the resulting graph is not split then we return no and several subgraphs as certificates: The C_4 or the $2K_2$ that was returned as a certificate of split non-membership, and the two or four certificates of split non-membership for each edge of the first certificate. If the input graph cannot be turned into split by removing any single edge of this forbidden subgraph, then definitely it cannot be made into split by removing any other edge, either. So there is no need to check for further forbidden subgraphs.

The running time of the algorithm is clearly linear, since we need to run a linear-time algorithm at most a constant number of times. In the case of a yes answer, the certificate is an edge e, and a split partition (or 3-partition) of the graph minus this edge. This certificate can be authenticated in $O(n+m)$ time. In the case of a no reply, we have a constant number of certificates of constant size, and we can check in $O(n)$ time that each of them is a forbidden induced subgraph. ∎

5.2 Certifying recognition of split+1v graphs in linear time

Observation 45 *A split graph with more than one vertex is a split+1v graph.*

Proof. Split graphs are hereditary, therefore a split graph minus any vertex is still split. ∎

Theorem 46 *There is a linear-time certifying algorithm for recognizing split+1v graphs, such that a certificate for membership can be authenticated in $O(n+m)$ time, and a certificate for non-membership can be authenticated in $O(n)$ time.*

Proof. First we run a certifying algorithm for recognizing split graphs. If the reply from this algorithm is yes, and the input graph contains more than one vertex, we return yes, and any vertex of the graph according to the proof of Observation 45 as certificate. If the reply from the split recognition algorithm is no, then we examine the certificate for split non-membership. The certificate is a C_5, a C_4 or a $2K_2$, then for each vertex v of this induced subgraph, we check whether removing this vertex results in a split graph. This can be done in $O(n+m)$ time with a certifying split recognition algorithm. If we conclude that the resulting graph is split, we return yes, and we return v and a split partition (or 3-partition) of the graph without vertex v. If we conclude that the resulting graph is not split then we return no and several subgraphs as certificates: The C_5, the C_4 or the $2K_2$ that was returned as a certificate of split non-membership, and the four or five certificates of split non-membership for each vertex of the first certificate. If the input graph cannot be turned into split by removing any single vertex of this forbidden subgraph, then definitely it cannot be made into split by removing any other vertex, either. So there is no need to check for further forbidden subgraphs.

The running time of the algorithm is clearly linear, since we need to run a linear-time algorithm at most a constant number of times. In the case of a yes answer, the certificate is a vertex v, and a split partition (or 3-partition) of the graph minus this vertex. This certificate can be authenticated in $O(n+m)$ time. In the case of a no reply, we have a constant number of certificates of constant size, and we can check in $O(n)$ time that each of them is a forbidden induced subgraph. ∎

5.3 Certifying recognition of split$-1e$ graphs in linear time

Observation 47 *A split graph that is not a complete graph, is a split$-1e$ graph.*

Proof. It follows from Observation 42 since split graphs are self-complementary. So in a split graph $G=(V,E)$ with split partition $V=I+K$, we can add any edge between K and I to obtain a new split graph with one more edge. If I is completely connected to K, then we can add any edge between two vertices $a,b \in I$, so that we obtain a split graph with one more edge and split partition $I'+K'$ where, $K'=K\cup\{a\}$ and $I'=I\setminus\{a\}$. ∎

Observation 48 *A split$-1e$ graph does not contain C_5 as an induced subgraph.*

Proof. Because the complement of G is split$+1e$ and the complement of a C_5 is a C_5. ∎

Theorem 49 *There is a linear-time certifying algorithm for recognizing split$-1e$ graphs, such that a certificate for membership can be authenticated in $O(n+m)$ time, and a certificate for non-membership can be authenticated in $O(n)$ time.*

Proof. Since split graphs are self complementary, we can derive an algorithm for recognizing split$-1e$, using the recognition algorithm for split$+1e$ on the complement of our input graph. Since the complement of a $2K_2$ is a C_4, and vice versa, a forbidden subgraph in the complement graph, is a subgraph in the input graph as well. Furthermore, removing one of the edges of a forbidden subgraph in the complement is equivalent to add a non edge to the complement of that forbidden subgraph in the input graph. Hence we have the following algorithm. First we run a certifying algorithm for recognizing split graphs. If the reply from this algorithm is yes, and the input graph is not complete, we return yes, and an edge not in the graph according to the proof of Observation 47 as certificate. If the reply from the split recognition algorithm is no, then we examine the certificate for split non-membership. If the certificate is a C_5, we return no, and the C_5 as our certificate. If the certificate is a C_4 or a $2K_2$, then for each non-edge e of this induced subgraph, we check whether adding this edge results in a split graph. This can be done in $O(n+m)$ time with a certifying split recognition algorithm. If we conclude that the resulting graph is split, we return yes, and we return e and a split partition (or 3-partition) of the graph with the edge e. If we conclude that the resulting graph is not split then we return no and several subgraphs as certificates: The C_4 or the $2K_2$ that was returned as a certificate of split non-membership, and the two or four certificates of split non-membership for each non-edge of the first certificate. If the input graph cannot be turned into split by adding any single edge to this forbidden subgraph, then definitely it cannot be made into split by adding any other edge, either. So there is no need to check for further forbidden subgraphs.

The running time of the algorithm is clearly linear, since we need to run a linear-time algorithm at most a constant number of times (a $2K_2$ has four non-edges, and a C_4 only two). In the case of a yes answer, the certificate is an edge e, and a split partition (or 3-partition) of the graph plus this edge. This certificate can be authenticated in $O(n+m)$ time. In the case of a no reply, we have a constant number of certificates of constant size, and we can check in $O(n)$ time that each of them is a forbidden induced subgraph. ∎

5.4 Minimum split completion of split+1v and split+1e graphs in linear time

Using the above results, and the dynamic operations described previously, we are able to show the following result.

Theorem 50 *Minimum split completion of split+1v and split+1e graphs can be computed in linear time.*

Proof. First we run a certifying algorithm for recognizing split+1v or split+1e graphs. If the reply is no we return an error message. If the reply is yes, then we take the edge e or the vertex v returned as part of the certificate, we remove this edge or vertex from the input graph G to obtain a graph G' which we know is split, and we use the dynamic operation for adding edge e or vertex v to G' using the option of minimum completion in the case of a no reply to the corresponding query. Each of the described tasks takes linear time, and hence we have linear time algorithms for minimum split completion of split+1v and split+1e graphs. ∎

5.5 Minimum split deletion of split−1e graphs in linear time

Theorem 51 *Minimum split deletion of split−1e graphs can be computed in linear time.*

Proof. First we run a certifying algorithm for recognizing split−1e graphs. If the reply is no we return an error message. If the reply is yes, then we take the non-edge e returned as part of the certificate, we add edge e to the input graph G to obtain a graph G' which we know is split, and we use the dynamic operation for deleting edge e from G' using the option of minimum deletion in the case of a no reply to the corresponding query. Each of the described tasks takes linear time, and hence we have a linear time algorithm for minimum split deletion of and split−1e graphs. ∎

6 Concluding remarks

We have given a completely dynamic algorithm for recognizing and maintaining split graphs. With the presented dynamic operations, we are also able to give new linear-time vertex incremental algorithms for split graphs matching the running time of existing non-incremental algorithms for the same purposes, and linear time algorithms for computing minimum split completions of split1 + e and split+1v graphs and minimum split deletions of split−1e graphs. All operations of the dynamic algorithm that is presented in this paper are either new, or match the running time of the operations proposed by Ibarra [12].

There is one more dynamic operation which we could have added to our algorithm: When we want to add a new vertex x to a given split graph $G = (V, E)$, and the reply from vertex addition query is no, we could have given an algorithm for computing a minimal split deletion (maximal split subgraph) of G_x. Actually, such a dynamic operation can easily be obtained from the results of Section 12 and can be implemented in $O(|N_x|)$ time, where N_x is the neighborhood of x in G_x. This would in turn result in a linear-time vertex incremental algorithm for computing a maximal split subgraph. We do not include such an operation for two reasons. To keep the length of the paper reasonable, and because minimal split deletions of arbitrary graphs are not as interesting as minimal split completions, since in many cases unless the input graph is very dense, a maximal split subgraph will consist of many isolated vertices.

References

[1] A. Berry, P. Heggernes, and Y. Villanger. A vertex incremental approach for maintaining chordality. *Discrete Mathematics.*, 306-3:318–336, 2006.

[2] C. Crespelle and C. Paul, Fully Dynamic Algorithm for Recognition and Modular Decomposition of Permutation Graphs. *Proceedings WG 2005, Springer Lecture Notes in Computer Science*, 3787:38–48, 2005.

[3] S. Földes and P. L. Hammer. Split graphs. *Congressus Numerantium*, 19:311–315, 1977.

[4] M. C. Golumbic. Algorithmic Graph Theory and Perfect Graphs. Second edition. Annals of Discrete Mathematics 57. Elsevier, 2004.

[5] P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

[6] P. Heggernes and F. Mancini, Minimal Split Completion of Graphs, *Proceedings LATIN 2006, Springer Lecture Notes in Computer Science*, 3887:592–604, 2006.

[7] P. Heggernes, F. Mancini, and C. Papadopoulos. Making arbitrary graphs transitively orientable: Minimal comparability completions. *Proceedings ISAAC 2006, Springer Lecture Notes in Computer Science*.

[8] P. Heggernes and D. Kratsch. Linear-time certifying algorithms for recognizing split graphs and related graph classes. Reports in Informatics 328, University of Bergen, Norway, June 2006.

[9] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. *Proceedings ESA 2005, Springer Lecture Notes in Computer Science*, 3669:403–414, 2005.

[10] P. Hell, R. Shamir, and R. Sharan, A fully dynamic algorithm for recognizing and representing proper interval graphs. *SIAM J. Comput.*, 31:289-305, 2002.

[11] L. Ibarra, Fully dynamic algorithms for chordal graphs. *Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA99)*, 923-924, 1999.

[12] L. Ibarra. Fully dynamic algorithms for chordal graphs and split graphs. Technical Report DCS-262-IR, University of Victoria, Canada, 2000.

[13] L. Ibarra. A fully dynamic algorithm for recognizing interval graphs using the clique-separator graph. Technical Report DCS-263-IR, University of Victoria, Canada, 2001.

[14] D. Kratsch, R. M. McConnell, K. Mehlhorn and J. P. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *Proc. 14th annual ACM-SIAM Symp. on Discrete Algorithms* (SODA 2003), 158–167, 2003.

[15] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Disc. Appl. Math.*, 113:109–128, 2001.

[16] S.D. Nikolopoulos and L. Palios, Adding an Edge in a Cograph. *Proceedings WG 2005, Springer Lecture Notes in Computer Science*, 3787:214-226, 2005.

[17] R. Shamir and R. Sharan, A fully dynamic algorithm for modular decomposition and recognition of cographs. *Discrete Appl. Math.*, 136:329-340, 2004.